我们专注于嵌入式

GQ2440
开发板用户手册

V1.2（20110120 版）

公司地址：成都市牛王庙东恒国际 1 栋 2 期 701

技术咨询 QQ：550491596　技术交流 QQ 群：65212116

公司网站: http://www.enjoylinux.cn 电话: 028-66501487

开发板购买地址: http://enjoylinux.taobao.com

目录

第 1 章 GQ2440 开发板介绍

1.1 GQ2440 开发板简介

GQ2440 是一款高性价比的 ARM9 开发板，它采用 Samsung S3C2440 微处理器，参考三星公版 smdk2410 优化设计而来。采用 6 层板设计，PCB 沉金工艺，并采用专业稳定的 CPU 内核电源芯片和复位芯片来保证系统运行时的稳定性。专业等长布线，保证关键信号线的信号完整性。生产采用机器贴片，专业大厂批量生产，出厂时都经过严格的质量控制。配合这本十分详细的手册，可以迅速帮你掌握嵌入式 Linux 和 WinCE 开发的流程，只要有 C 语言基础的人一般 2 周即可入门。GQ2440 与同类开发板相比有如下优点:

1.采用核心板+底板形式，核心板采用 6 层板设计，更稳定运行，方便单位用户开发;

2.在合适的位置预留摄像头等重要接口，方便客户设计使用;

3.串口接口增加了 2 个 DB9 接口，方便客户应用在 3 串口场合; 更有 3 个 TTL 电平 com 口外引;

4.同时包含音频输入和输出接口便于相关开放;

5.4 个 USB 主口，更加方便扩展其他模块。

1.1.1 GQ2440 开发板硬件资源特性

CPU 处理器

- Samsung S3C2440A，主频400MHz，最高533Mhz

SDRAM 内存

- 在板64M SDRAM
- 32bit 数据总线
- SDRAM 时钟频率高达100MHz

FLASH 存储

在板 256M/1GB Nand Flash, 掉电非易失(用户可定制64M/128M/256M/512M/1G)

在板2M Nor Flash，掉电非易失，已经安装BIOS

LCD 显示

- 板上集成4 线电阻式触摸屏接口，可以直接连接四线电阻触摸屏
- 支持黑白、4 级灰度、16 级灰度、256 色、4096 色STN 液晶屏，尺寸从3.5 寸到12.1 寸，屏幕分辨率可
以达到1024x768 象素;
- 支持黑白、4 级灰度、16 级灰度、256 色、64K 色、真彩色TFT 液晶屏，尺寸从3.5 寸到12.1 寸，屏幕
分辨率可以达到1024x768 象素;
- 标准配置为统宝 3.5” 真彩 LCD，分别率 320x240，带触摸屏;

接口和资源

- 1个100M 以太网RJ-45 接口(采用DM9000 网络芯片)
- 3个串行口
- 4个USB Host
- 1个USB Slave B 型接口
- 1个SD 卡存储接口
- 1路立体声音频输出接口，一路麦克风接口;
- 1个2.0mm 间距10 针JTAG 接口
- 4 USER Leds
- 6 USER buttons(带引出座)
- 1个PWM 控制蜂鸣器
- 1个可调电阻，用于AD模数转换测试
- 1个I2C 总线AT24C08 芯片，用于I2C 总线测试
- 1个2.0 mm 间距20pin 摄像头接口
- 板载实时时钟电池
- 电源接口(5V)，带电源开关和指示灯

系统时钟源

- 12M 无源晶振

实时时钟

- 内部实时时钟（带后备锂电池）

扩展接口

- 1个54 pin 2.0mm GPIO及系统数据总线接口
- 1个20 pin 2.0mm GPIO及系统地址总线接口

规格尺寸

- 115×165(mm)

操作系统支持

- Linux2.6.32.2 + Qtopia-2.2.0
- WindowsCE.NET 6.0(R3)

1.1.2 Linux 系统特性

版本

- Linux 2.6.32.2(BSP 可自适应64M/128M/256M/512M/1GB Nand Flash)

支持的文件系统

- yaffs2(可读写的文件系统，推荐使用)
- cramfs(压缩的只读文件系统，不在线更新数据时推荐使用)
- Ext2
- Fat32
- NFS(网络文件系统，开发驱动程序及应用程序时方便使用)

基本驱动程序(以下驱动均以源代码方式提供)

- 3串口标准驱动
- DM9000 驱动程序
- 音频驱动(UDA1341)(可录音)
- RTC 驱动(可掉电保存时间)
- 用户LED 灯驱动
- USB Host 驱动
- 真彩LCD 驱动(含1024x768VGA 驱动)
- 触摸屏驱动
- 免驱的万能USB 摄像头驱动
- USB 鼠标、USB 键盘驱动、优盘、移动硬盘
- SD 卡驱动，可支持高速SD 卡，最大容量可达32G
- I2C-EEPROM
- PWM 控制蜂鸣器
- LCD 背光驱动
- A/D 转换驱动

- 看门狗驱动(看门狗复位相当于冷复位)

Linux 应用及服务程序

- busybox1.13(Linux 工具集，包含常用Linux 命令等)
- Telnet、Ftp、inetd(网络远程登录工具及服务)
- boa(web server)
- madplay(基于控制台的mp3 播放器)
- snapshot(基于控制台的抓图软件)
- ifconfig、ping、route 等(常用网络工具命令)

嵌入式图形系统平台(以源代码方式提供)

- Qt/Embedded 2.2

分为x86 和arm 两个版本

实用的 Qtopia 测试程序

- A/D 转换测试
- LED 控制
- Buttons 按键测试
- I2C-EEPROM 读写测试
- LCD 测试
- Ping 测试
- 万能免驱USB 摄像头动态预览并拍照
- 录音机
- Web 浏览器
- 看门狗测试
- 网络设置(可保存参数)
- 背光控制
- 语言设置：可设置中英文
- 随手写：主要用于测试触摸笔的准确性
- MMC/SD 卡和优盘自动挂载和卸载

1.1.3 WindowsCE 5.0 系统特性

版本

- WindowsCE 5.0 (BSP 可自适应64M/128M/256M/512M/1G Nand Flash)

特性

- 支持.NET 2.0
- 支持SQL Mobile
- 支持注册表保存
- 支持快速开机启动(10 秒以内)
- 提供了目前国内最完善的WindowsCE 5.0 BSP(含bootloader)，并100%开放源代码，包括:

1 - PWM 控制蜂鸣器
2 - CMOS 摄像头
3 - I2C-EEPROM 读写
4 - ADC 模数转换
5 - 看门狗
6 - 用户按键(6 个)
7 - 用户LED(4 个)
8 - 背光控制和管理(可设定背光关闭时间,并通过按键，触摸屏，键盘，鼠标唤醒)
9 - RTC
10 - DM9000 网卡
11- 大容量高速SD 卡，最高可达32G
12- 优盘、USB 键盘、USB 鼠标等
13- 音频播放和录音
14- 触摸屏
15-LCD 驱动(支持N35/T35, A70, L80, VGA1024x768，分别对应分辨率:240x320, 800x480, 640x480, 1024x768)
16- 通过简单修改头文件定义即可支持多种型号的LCD(保持和WinCE5 一致)
17- LCD 旋转设置
18- 完善的3 个串口驱动(通过简单修改头文件定义可以指定UART0 作为DEBUG 输出或者普通串口，保持和WinCE5 一致)
19- 通过修改Nboot 头文件可以方便的自定义进度条的颜色、位置、长宽，以及开机图片的位置、背景
20- 增加了方便的Logo 制作工具StartLogoMaker(绿色软件，可运行于XP 或者Vista, Win7)
21- 通过USB 可以烧写普通的bmp 文件作为开机画面

配合以上全新完善的 BSP，增加了以下各种WindowsCE 实用小程序，基本和Qtopia Apps 保持相同或类似的界面：
- (1) LCD-Test: LCD 测试
- (2) PWM-Buzzer: PWM 控制蜂鸣器
- (3) CMOS Camera: CMOS 摄像头动态预览并拍照
- (4) I2C-EEPROM: 读写基于I2C 总线的EEPROM 测试
- (5) Watchdog: 看门狗测试
- (6) AD-Convert: ADC 转换测试
- (7) Buttons: 按键测试
- (8) Rotate: 屏幕旋转设定(可保存旋转结果)
- (9) Autorun-Setting: 设定开机自动运行程序
- (11) Recorder: 录音测试 (原有,界面稍做调整)，提供测试源代码
- (12) 串口助手：提供测试源代码
- (13) LED 测试：提供测试源代码

1.1.4 WindowsCE 6.0 系统特性

版本
- WindowsCE Embedded 6.0 (BSP 可自适应64M/128M/256M/512M/1G Nand Flash)

特性
- 基于最新的补丁Release3(2009.10 发布，用户可自行添加移动版QQ 组件，比较大)
- 支持.NET 3.5
- 支持全盘目录可读写，可以安装更多第三方软件，如蒙恬手写输入法
- 支持快速开机启动(10 秒以内)
- 提供了目前国内最完善的WindowsCE 6.0 BSP(含bootloader)，并100%开放源代码，包括：

1 - PWM 控制蜂鸣器
2 - CMOS 摄像头
3 - I2C-EEPROM 读写
4 - ADC 模数转换
5 - 看门狗
6 - 用户按键(6 个)

7 - 用户LED(4 个)
8 - 背光控制和管理(可设定背光关闭时间,并通过按键，触摸屏，键盘，鼠标唤醒)
9 - RTC
10 - DM9000 网卡
11- 大容量高速SD 卡，最高可达32G
12- 优盘、USB 键盘、USB 鼠标等
13- 音频播放和录音
14- 触摸屏
15-LCD 驱动(支持N35/T35, A70, L80, VGA1024x768，分别对应分辨率:240x320, 800x480, 640x480, 1024x768)
16- 通过简单修改头文件定义即可支持多种型号的LCD(保持和WinCE5 一致)
17- LCD 旋转设置
18- 完善的3 个串口驱动(通过简单修改头文件定义可以指定UART0 作为DEBUG 输出或者普通串口，保持和WinCE5 一致)
19- 通过修改Nboot 头文件可以方便的自定义进度条的颜色、位置、长宽，以及开机图片的位置、背景
20- 增加了方便的Logo 制作工具StartLogoMaker(绿色软件，可运行于XP 或者Vista, Win7)
21- 通过USB 可以烧写普通的bmp 文件作为开机画面
配合以上全新完善的 BSP，增加了以下各种WindowsCE 实用小程序，基本和Qtopia Apps 保持相同或类似的界面:
- (1) LCD-Test: LCD 测试
- (2) PWM-Buzzer: PWM 控制蜂鸣器
- (3) CMOS Camera: CMOS 摄像头动态预览并拍照
- (4) I2C-EEPROM: 读写基于I2C 总线的EEPROM 测试
- (5) Watchdog: 看门狗测试
- (6) AD-Convert: ADC 转换测试
- (7) Buttons: 按键测试
- (8) Rotate: 屏幕旋转设定(可保存旋转结果)
- (9) Autorun-Setting: 设定开机自动运行程序
- (11) Recorder: 录音测试 (原有,界面稍做调整)，提供测试源代码
- (12) 串口助手: 提供测试源代码

- (13) LED 测试：提供测试源代码

1.1.5 资料光盘资源说明

本开发板提供一张 DVD 资料光盘，主要有以下内容：

(1) Linux 源代码和工具

包括：

- 统一的交叉编译器 arm-linux-gcc-4.3.2 with EABI
- 最新的 Linux 内核源代码包Linux-2.6.32.2，包含十分齐全完善的BSP
- 嵌入式图形界面 Qte/Qtopia 源代码包：x86-qtopia 和arm-qtopia，分别对应PC 版本和ARM 版本，内含编译脚本
- Busybox 源代码包及缺省配置文件
- Bootloader 源代码(vboot)：使用arm-linux-gcc 交叉编译器编译
- Linux 编程示例：含串口，PWM，AD，EEPROM，多线程等

(2) WinCE 源代码和软件

包括：

- WindowsCE 5.0/6.0 的BSP，是目前2440 开发板中最完善的BSP，100%开放源代码
- Bootloader 源代码(nboot)：使用ADS 可以编译，WindowsCE 5.0/6.0 共用相同的bootloader，均可实现快速启动，快速显示开机Logo，支持2M 以内的24-bit 真彩bmp 图片(16-bit 的1024x768 的bmp 图片大小为1.5M)
- 基于 WindowsCE 5.0/6.0 制作的SDK，这是2 个不同版本的SDK
- GQ2440.pbxml：内核示例工程，汇集了最常用的组件功能，用户可以在此基础上增删适合自己需要的内核工程

(3) Bootloader源代码

u-boot_2009_11及vivi源代码

(4)非操作系统示例代码

包括GQ2440_Test和一些ADS示例工程，GQ2440_Test是由三星原厂的 2440test 改进而来的一个裸机测试程序，使用ADS 编译，在此仅为用户学习参考使用。

测试项目包括：中断方式按键测试，RTC 实时时钟测试，ADC 数模转换测试，IIS 音频播放wav 测试，IIS 音频录音测试，触摸屏测试，I2C 总线读写AT24C08 测试，三星3.5”LCD、640x480真彩液晶测试，LCD2VGA 输出测试等。

(5)Windows 平台下的实用工具软件

为了方便国内用户更加方便的使用开发板套件,我们制作和搜集了一些常用的Windows 平台的工具软件，并把它们统一放在光盘的“windows 平台工具”目录中，包括:

- ADS 1.2 集成开发环境安装程序
- H-JTAG: 支持烧写各种NOR Flash
- Vmware: 著名的虚拟机软件，版本VMware-workstation-6.5.1
- Dnw: 用来实现USB 下载
- ActiveSync: 微软的PC 同步软件
- Usb 下载驱动: 开发板BIOS 模式下所需的PC 端驱动程序
- LCD彩色图片转换工具BMP_to_H: 将图片转化为RGB位图格式

(6)各种数据手册和原理图

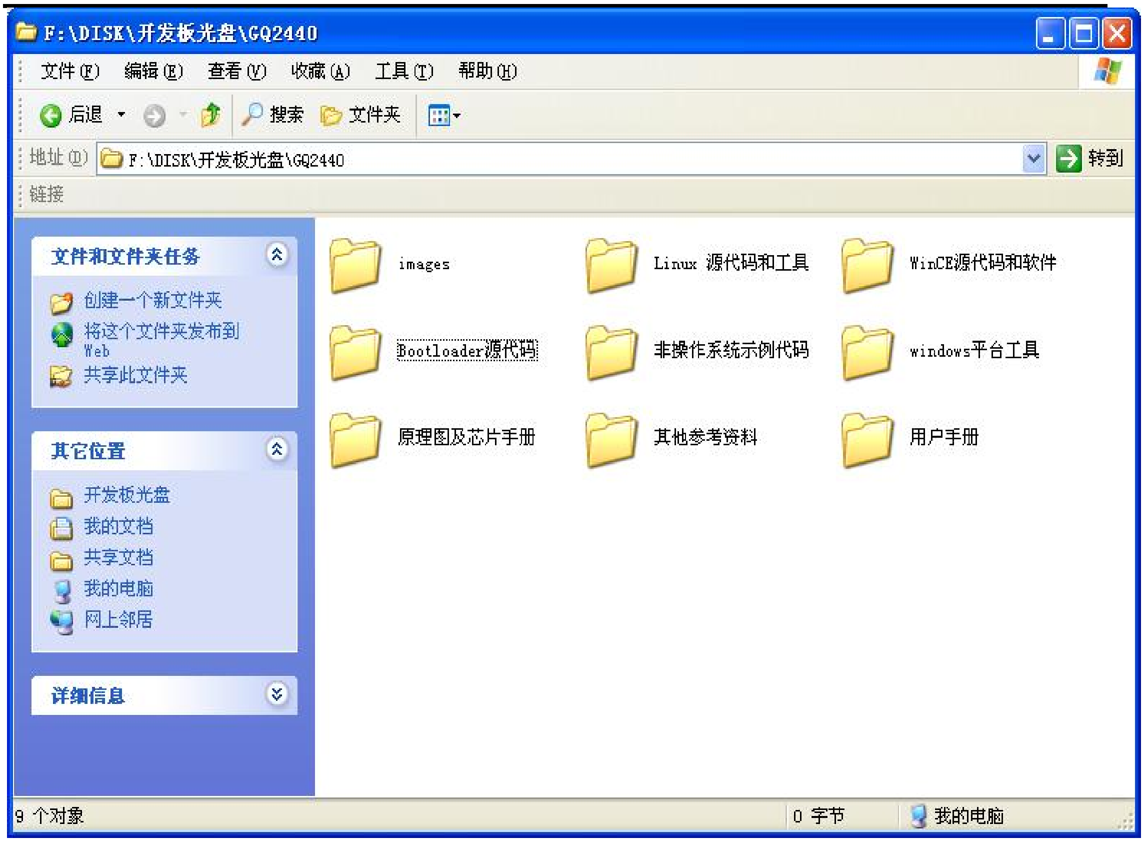
包含开发板的原理图(分为pdf 格式和protel99se 格式)及封装库，开发板上各个芯片的数据手册，开发板配套的LCD 数据手册，以及LCD 驱动板原理图等。

(7)images: 主要存放是linux和wince的安装镜像文件。

(8)其他参考资料: 为了方便用户学习使用开发板，我们还不定期在网上搜集一些和学习开发板相关的资料，也有一些是由网友无私提供的。

(9)用户手册

开发板的用户手册，用户可以在我们网站下载最新版本的用户手册。



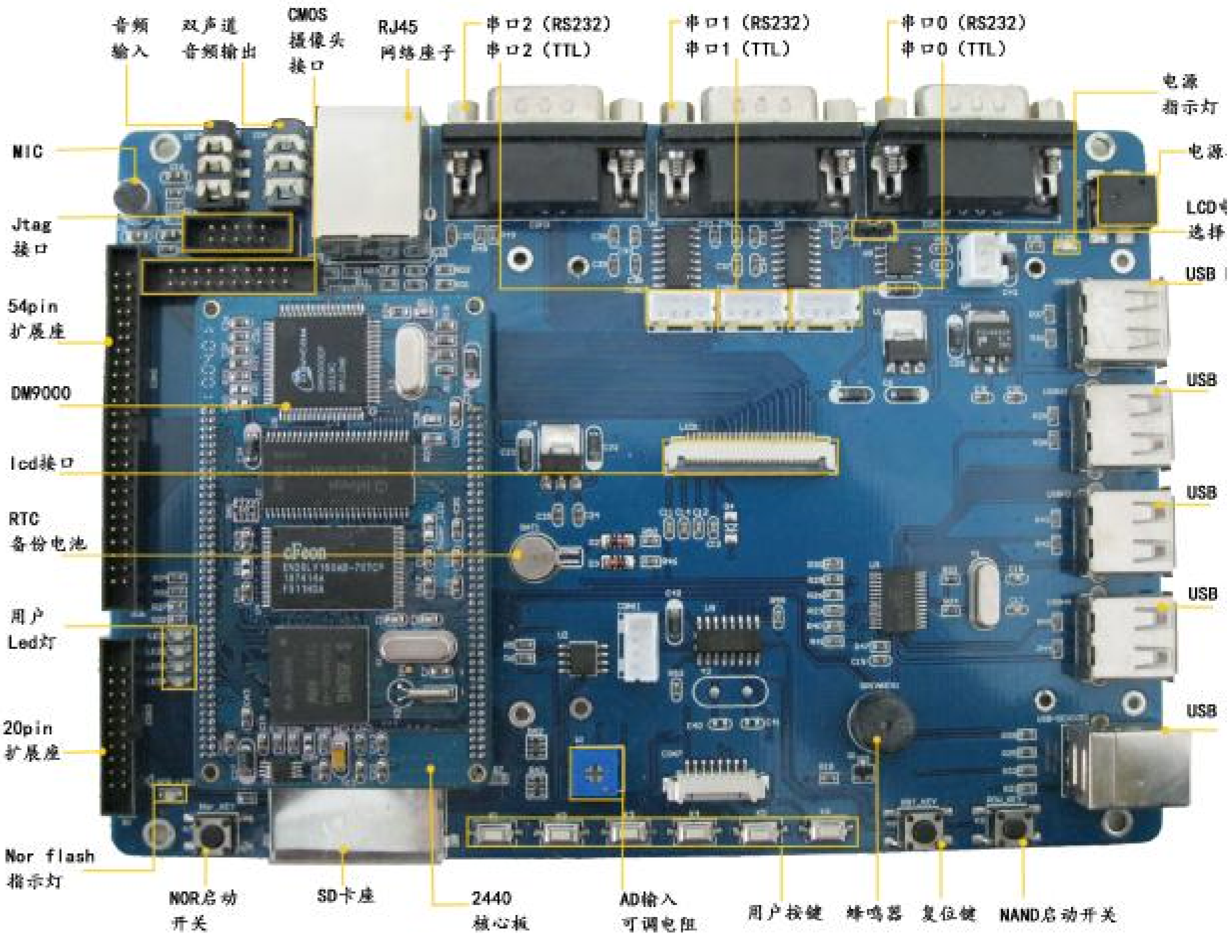
1.2 接口布局及跳线

1.2.1 跳线说明

开发板上只有一个跳线 J3，它用于选择LCD 驱动板的输入电压，在标准配置中，所接为3.5寸LCD，电压选择为3.3V，当接为7寸LCD时应该选择5V电压。

1.2.2 接口布局

GQ2440 接口布局如下图所示，它在比较紧凑的165 x 115mm 面积上精致安排了开发者常用的各种常用接口，并且还引出了供开发测试需要的富余的IO口和总线接口：

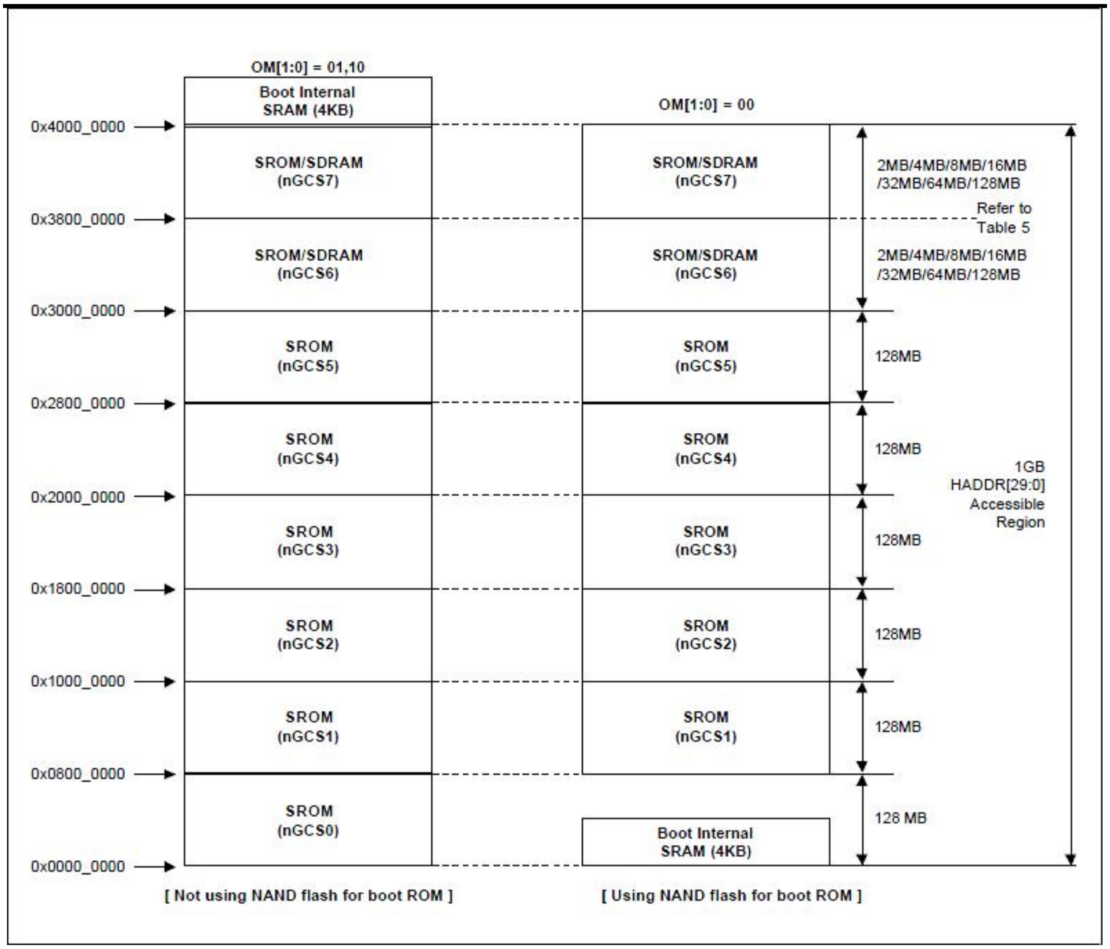


1.3 接口资源说明

本小节详细介绍了开发板上每个接口或模块的引脚定义和占用的 CPU 资源，光盘中还有本开发板的完整原理图和封装库(分为 pdf 格式和 Protel99SE 格式),以供开发板参考使用。

1.3.1 地址空间分配和片选信号定义

S3C2440 支持两种启动模式:一种是从 Nand Flash 启动;一种是从 Nor Flash 启动。在此两种启动模式下，各个片选的存储空间分配是不同的，如下图:



上图中，左边是 nGCS0 片选的 Nor Flash 启动模式下的存储分配图；右边是 Nand Flash 启动模式下的存储分配图；SFR Area 为特殊寄存器地址控制。

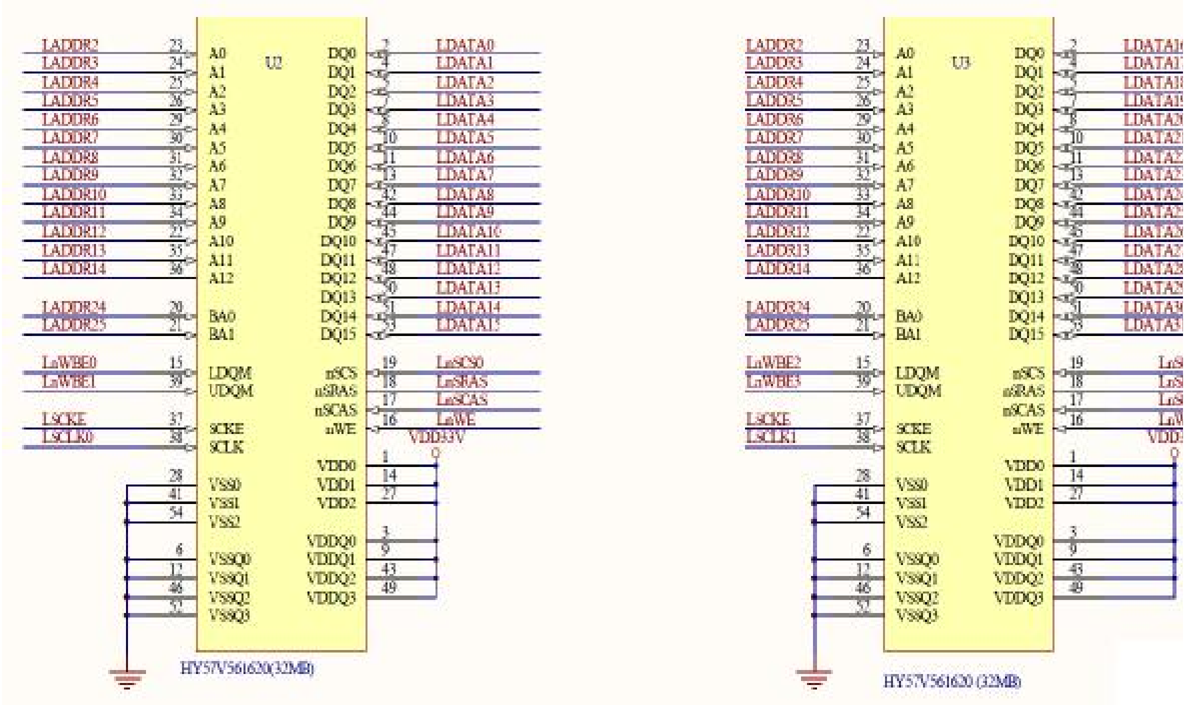
下面是器件地址空间分配和其片选定义在进行器件地址说明之前，有一点需要注意，nGCS0 片选的空间在不同的启动模式下，映射的器件是不一样的。由上图可以知道：

- 在 NAND Flash 启动模式下，内部的 4K Bytes BootSram 被映射到 nGCS0 片选的空间；
- 在 Nor Flash 启动模式下(非 Nand Flash 启动模式)，与 nGCS0 相连的外部存储器 Nor Flash 就被映射到 nGCS0 片选的空间。

SDRAM 地址空间：0x30000000 ~ 0x34000000。

1.3.2 SDRAM 存储系统

GQ2440 使用了两片外接的 32M bytes 总共 64M bytes 的 SDRAM 芯片(型号为：HY57V561620FTP/MT48LC16M16A2)，一般称之为内存，它们并接在一起形成 32-bit 的总线
数据宽度，这样可以增加访问的速度；因为是并接，故它们都使用了 nGCS6 作为片选，根据 CPU 手册 5-2 中的介绍可知，这就决定了它们的物理起始地址为 0x30000000，下面是摘自 GQ2440 原理图中的 SDRAM 部分原理图。



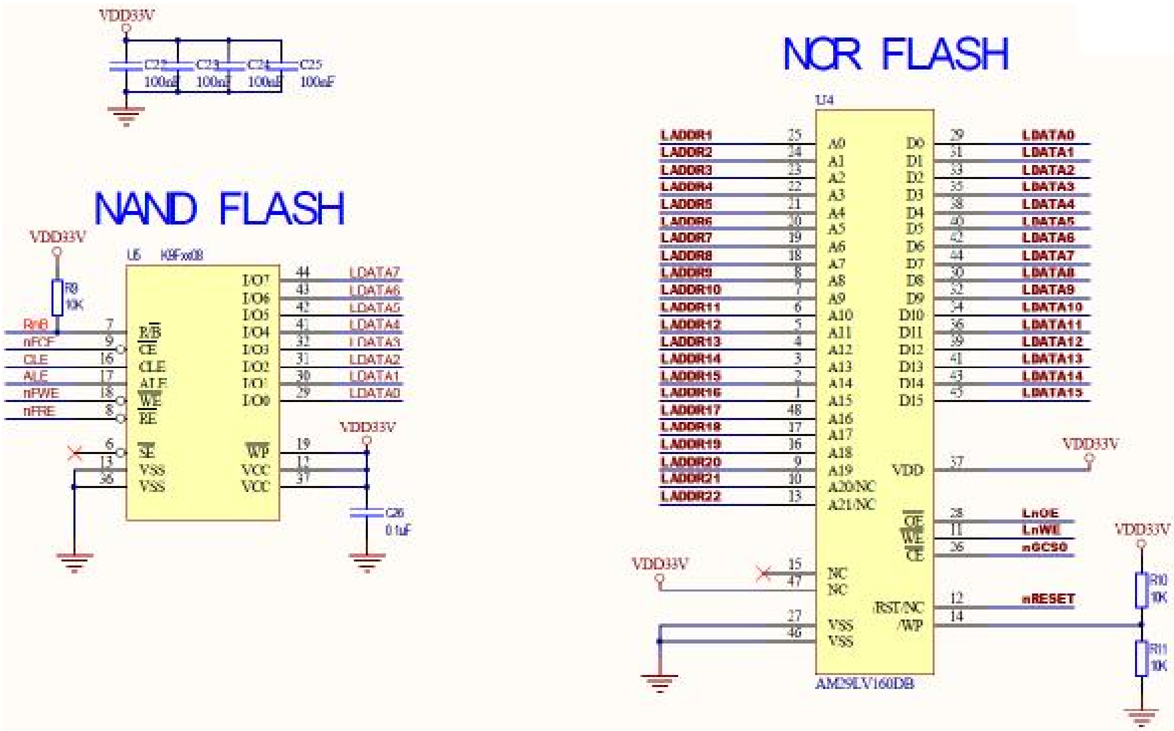
1.3.3 FLASH 存储系统

GQ2440 具备两种 Flash，一种是 Nor Flash，型号为 EN29LV160AB(AMD29LV160DB

与此引脚兼容)，大小为 2Mbyte; 另一种是 Nand Flash，型号为 K9F1G08,大小为 128M(旧版本为 K9F1208，大小为 64Mbyte)。S3C2440 支持这两种 Flash 启动系统，在 GQ2440 中通过 Nor_KEY 和 POW_KEY 按键，你可以选择从 NOR 还是从 NAND 启动系统，如右图所示。实际的产品中大都使用一片 Nand Flash 就够了，因为我们为了方便用户开发学习，因此还保留了 Nor Flash。

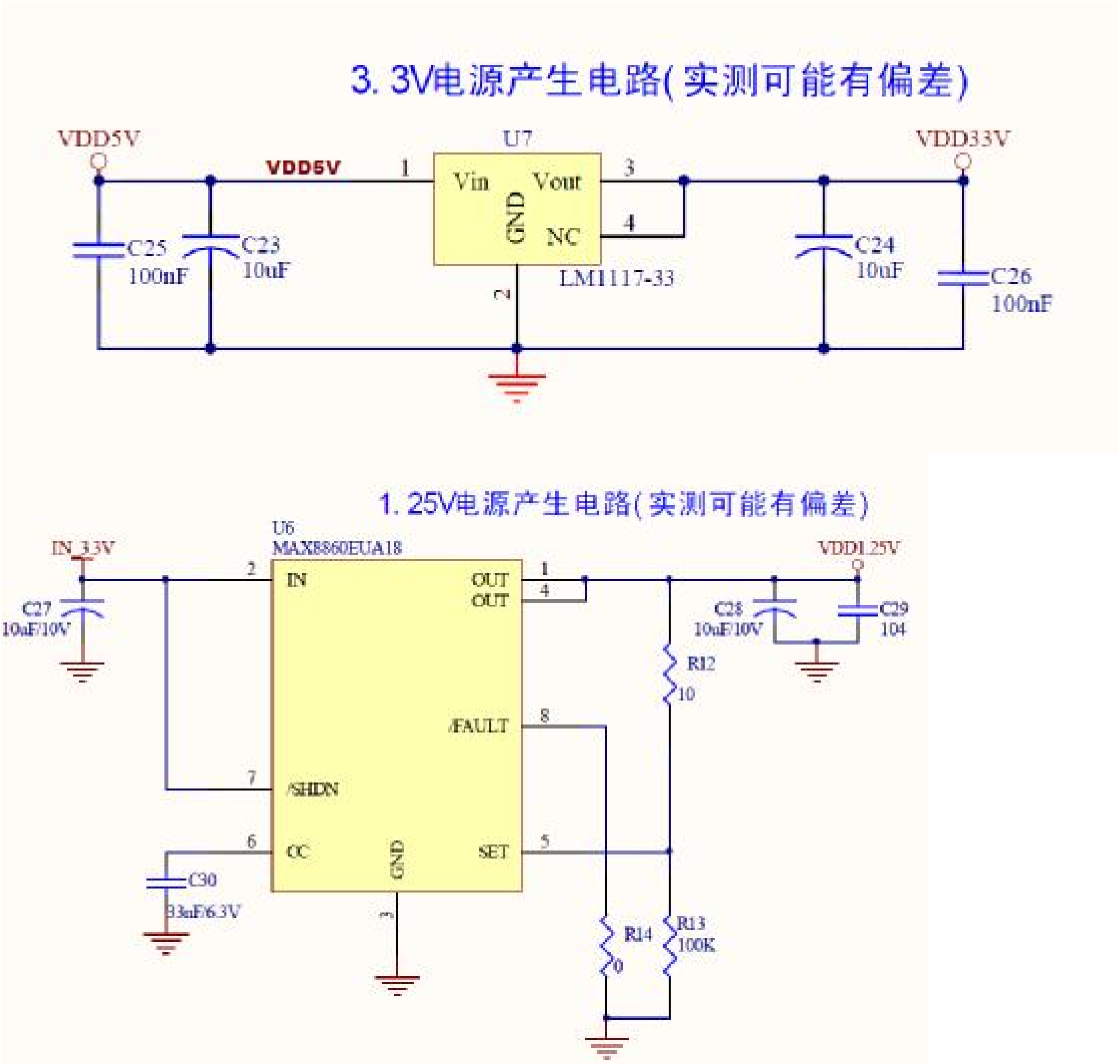
Nand Flash 不具有地址线，它有专门的控制接口与 CPU 相连，数据总线为 8-bit，但这并不意味着 Nand Flash 读写数据会很慢。大部分的优盘或者 SD 卡等都是 Nand Flash 制成的设备。

从下面的原理图可以看出，Nor Flash 采用了 A1-A22 总共 22 条地址总线和 16 条数据总线与 CPU 连接，请注意地址是从 A1 开始的，这意味着它每次最小的读写单位 2-byte，因此根据原理图，该设计总共可以兼容支持最大 8Mbyte 的 Nor Flash，实际我们的开发板上只用了 A1-A20 条地址线，因为与 A21、A22 相连的 EN29LV160AB 的相应引脚是悬空的。



1.3.4 电源系统及接口

本开发板的电源系统比较简单，直接使用外接的 5V 电源，通过降压芯片产生整个系统所需要的三种电压：3.3V、1.8V、1.25V。

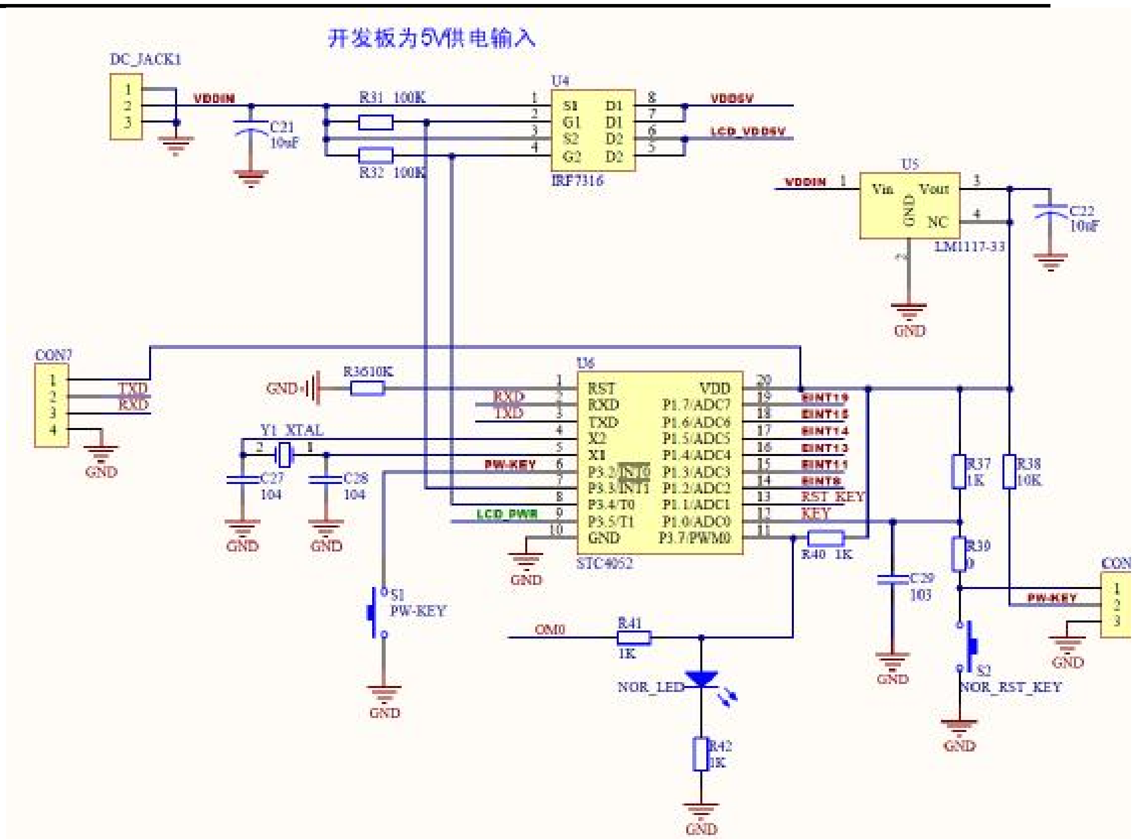


为了方便用户选择从 Nor Flash 或者 Nand Flash 启动，我们设计了如下图所示的启动电路。在未给开发板上电的时候，通过 PW_KEY 和 NOR_RST_KEY（分别对应右图中的 POW_KEY 和 Nor_KEY）可以使开发板分别从 Nand Flash 和 Nor Flash 启动，再次按下对应的键则可关机。两个按键具体关系如下：

操作前状态	操作	操作后状态
未上电	按下 POW_KEY	从 Nand Flash 启动
未上电	按下 Nor_KEY	从 Nor Flash 启动
正从 Nor 启动	按下 POW_KEY	关机
	按下 Nor_KEY	关机
正从 Nand 启动	按下 POW_KEY	关机
	按下 Nor_KEY	从 Nor Flash 启动



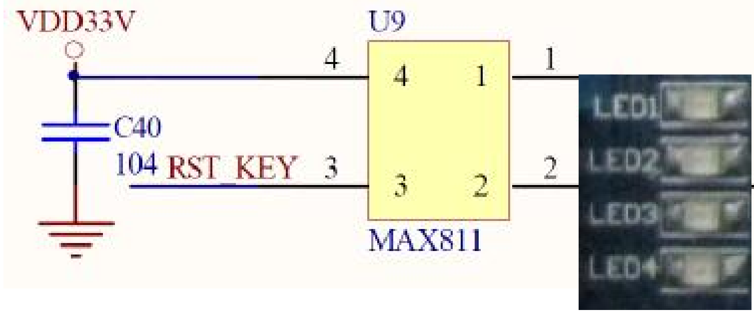


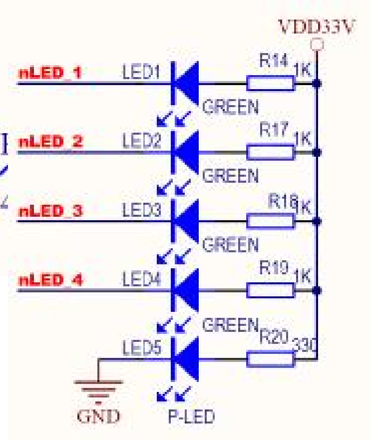


1.3.5 复位系统

本开发板采用专业的复位芯片 MAX811 实现 CPU 所需要的低电平复位，见下图：







1.3.6 用户 LED

LED 是开发中最常用的状态指示设备，本开发板具有 4 个用户可编程 LED，它们直接与 CPU 的 GPIO 相连接，低电平有效(点亮)，详细的资源占用如下表:

	LED1	LED2	LED3	LED4
GPIO	GPB5	GPB6	GPB7	GPB8
可复用位	nXBACK	nXREQ	nXDACK1	nDREQ1
在原理图中的网络名	nLED_1	nLED_2	nLED_3	nLED_4

1.3.7 用户按键

本开发板总共有 6 个用户测试用按键，它们均从 CPU 中断引脚直接引出，属于低电平触发，这些引脚也可以复用为 GPIO 和特殊功能口，为了用户把它们引出作为其他用途，这 6 个引脚也通过 J4(开发板上丝印为 CON2) 引出，6 个按键和 J4 的定义如下：

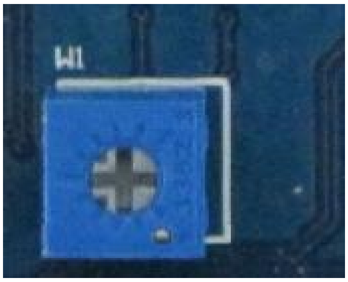


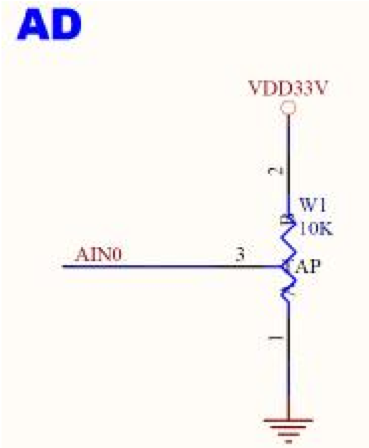
	K1	K2	K3	K4	K5	K6
中断	EINT8	EINT11	EINT13	EINT14	EINT15	EINT19
对应 GPIO	GPG0	GPG3	GPG5	GPG6	GPG7	GPG11
特殊功能口	无	nSS1	SPIMISO 1	SPIMOSI 1	SPICLK1	TCLK1
J4 对应引脚	18	10	14	9	5	11

1.3.8 A/D 输入测试

本开发板总共可以引出 4 路 A/D(模数转换)转换通道，为了方便测试，AIN0 连接到了开发板上的可调电阻 W1，原理图如下图所示。

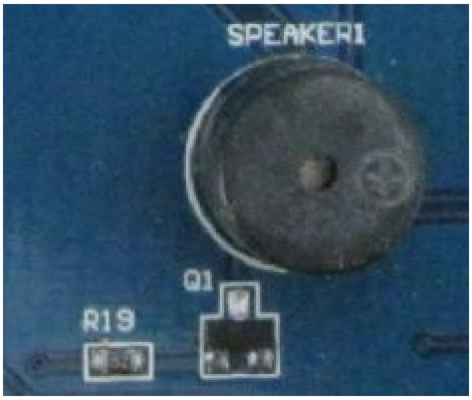


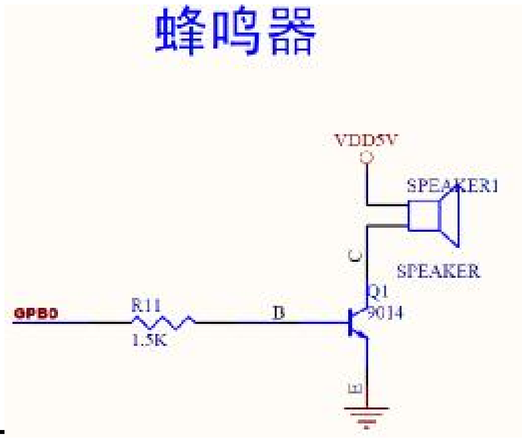




1.3.9 PWM 控制蜂鸣器

开发板的蜂鸣器 SPEAKER 是通过 PWM 控制的，原理图如下所示，其中 GPB0 可通过软件设置为 PWM 输出。

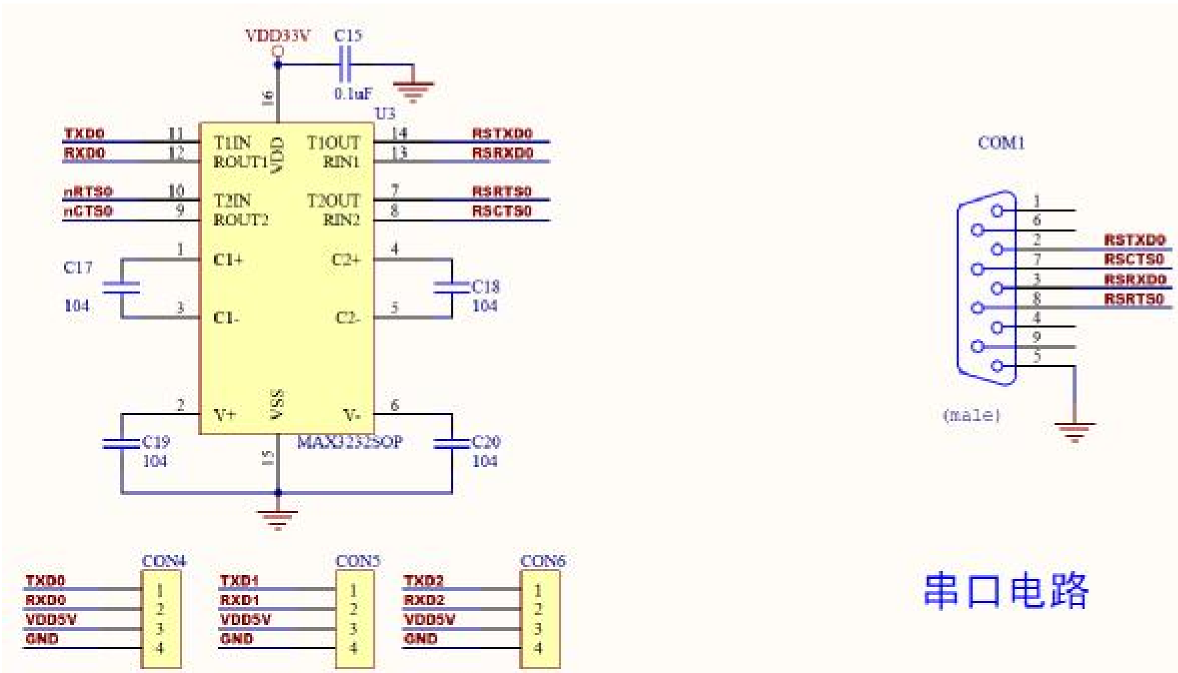




1.3.10 串口



S3C2440 本身总共有 3 个串口 UART0、1、2，其中 UART0,1 可组合为一个全功能的串口，在大部分的应用中，我们只用到 3 个简单的串口功能(本开发板提供的 Linux 和 WinCE 驱动也是这样设置的)，即通常所说的发送(TXD)和接收(RXD)，它们分别对应原理图上的 CON4、CON5、CON6（开发板丝印分别是 CON8、CON9、CON10），这 3 个接口都是从 CPU 直接引出的，是 TTL 电平。为了方便用户使用，UART0、UART1、UART2 都做了 RS232 电平转换，可以通过附带的直连线与 PC 机互相通讯。

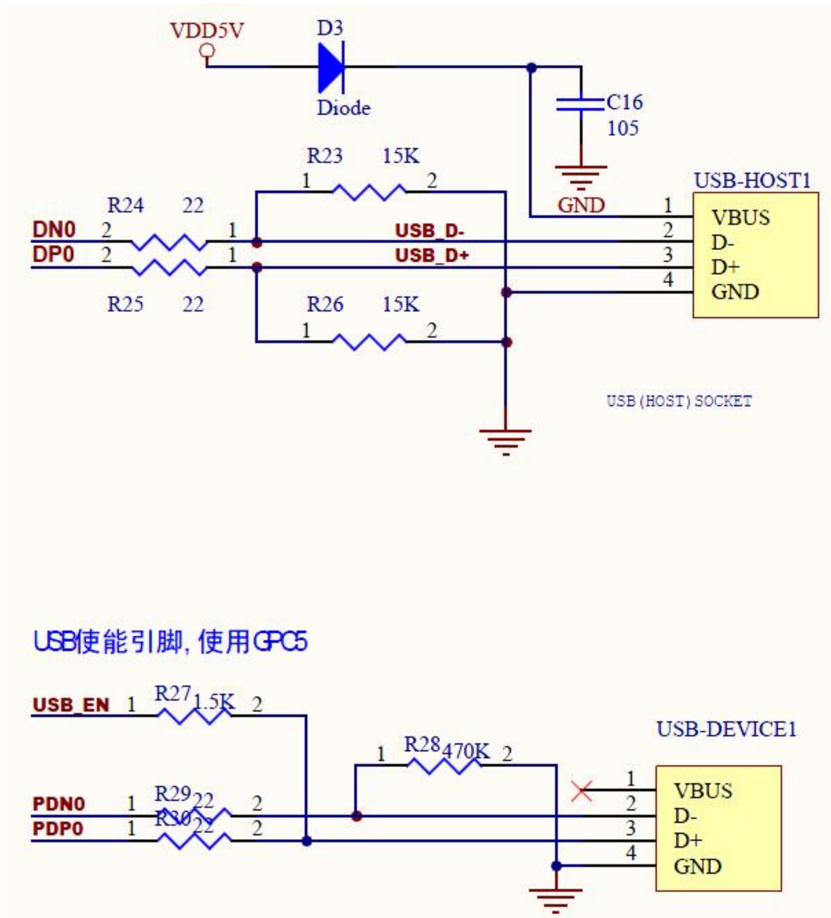


串口电路

1.3.11 USB 接口

本开发板具有两种 USB 接口，一个是 USB Host，它和普通 PC 的 USB 接口是一样的，
可以接 USB 摄像头、USB 键盘、USB 鼠标、优盘等常见的 USB 外设，另外一种是 USB Slave，
我们一般使用它来下载程序到目标板，当开发板装载了 WinCE 系统时，它可以通过 ActiveSync 软件和 Windows 系统进行同步，当开发板装载了 Linux 系统时，目前尚无相应的驱动和应用。为了方便用户通过程序控制 USB Slave 和 PC 的通断，我们设置了 USB_EN 信号，如图，它使用的 CPU 资源为 GPC5。





1.3.12 LCD 接口

本开发板的 LCD 接口是一个 41Pin 0.5mm 间距的白色座，其中包含了常见 LCD 所用的大部分控制信号(行场扫描、时钟和使能等)，和完整的 RGB 数据信号(RGB 输出为 8: 8: 8，即最高可支持 1600 万色的 LCD)；为了用户方便试验，还引出了 PWM 输出(GPB1 可通过寄存器配置为 PWM)， 和复位信号(nRESET)，其中 LCD_PWR 是背光控制信号。

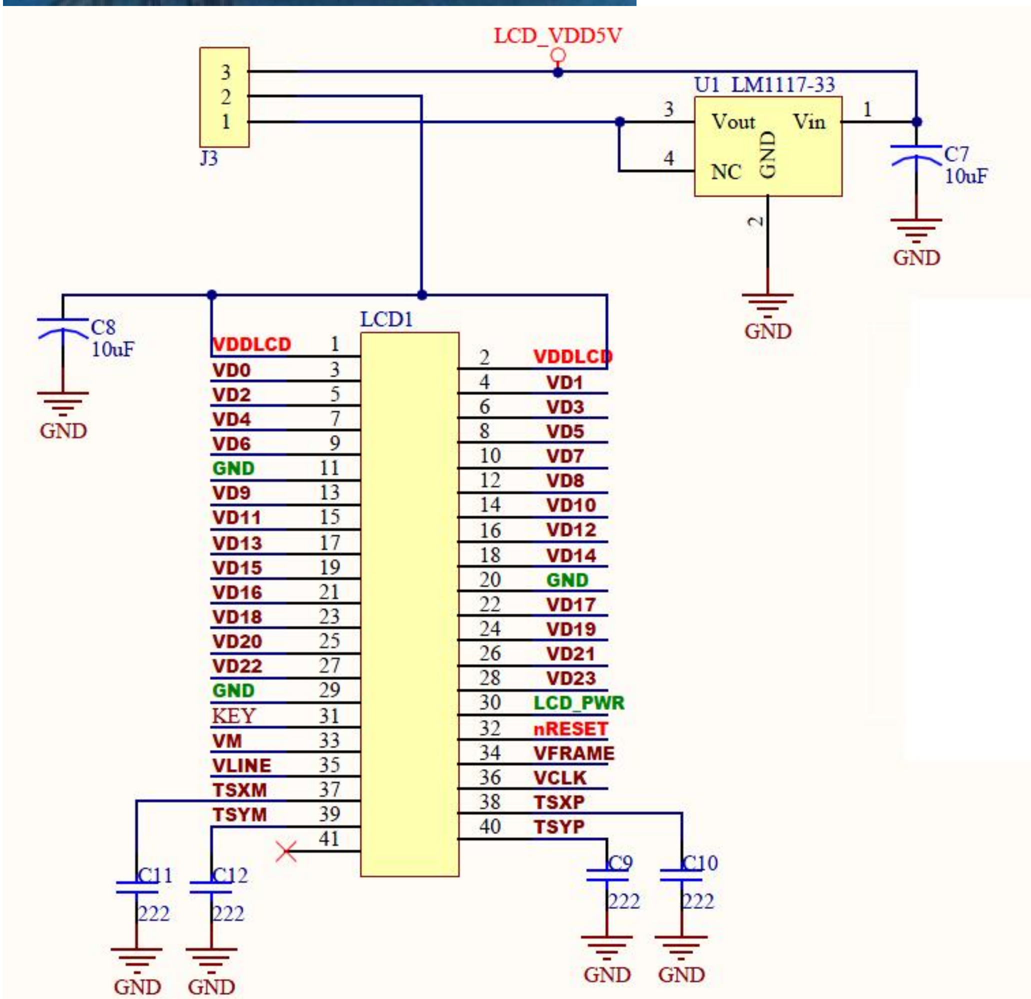
另外，37、38、39、40 为四线触摸屏接口，它们可以直接连接触摸屏使用。

图中的 J3 为 LCD 驱动板供电选择信号，目前我们的驱动板 3.5 寸屏使用 3.3V 供电，

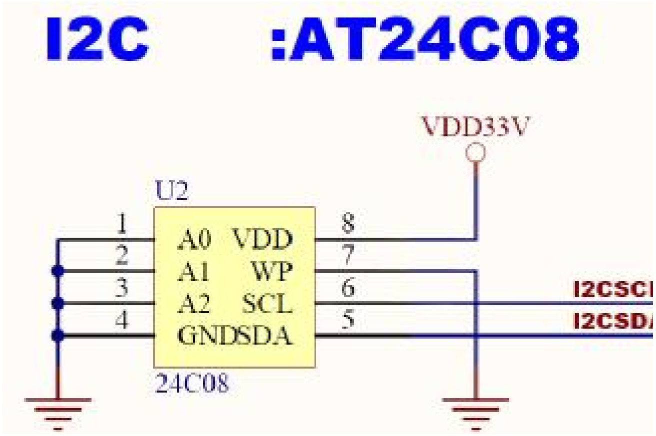
7 寸屏选择 5V 供电。


LCD1

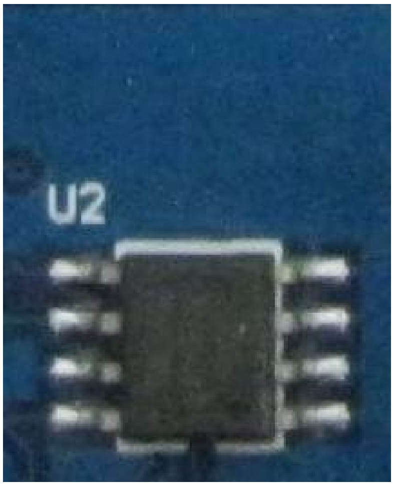

C26
3.3V
J3
5V


LCD_VDD5V
J3
3
2
1
U1 LM1117-33
Vout
Vin
NC
GND
C7
10uF
GND
C8
10uF
LCD1
VDDLCD 1
VD0 3
VD2 5
VD4 7
VD6 9
GND 11
VD9 13
VD11 15
VD13 17
VD15 19
VD16 21
VD18 23
VD20 25
VD22 27
GND 29
KEY 31
VM 33
VLINE 35
TSXM 37
TSYM 39
41
2 VDDLCD
4 VD1
6 VD3
8 VD5
10 VD7
12 VD8
14 VD10
16 VD12
18 VD14
20 GND
22 VD17
24 VD19
26 VD21
28 VD23
30 LCD_PWR
32 nRESET
34 VFRAME
36 VCLK
38 TSXP
40 TSYP
C11
222
C12
222
C9
222
C10
222
GND

1.3.13 EEPROM



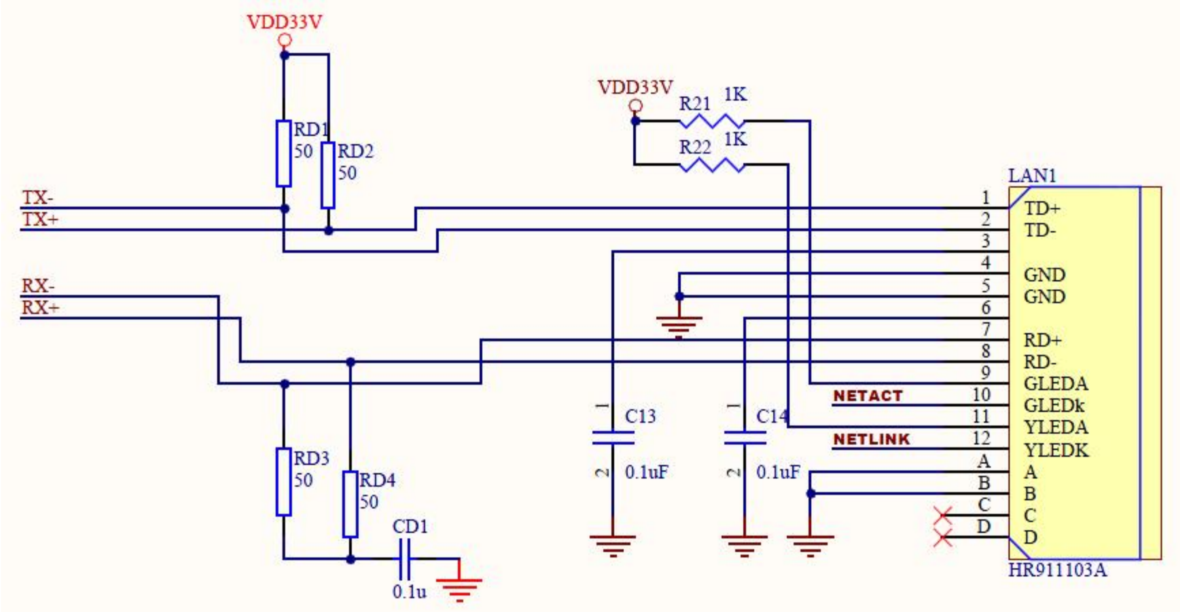
本开发板具有一个直接连接 CPU 之 I2C 信号引脚的 EEPROM 芯片 AT24C08，它的容量有 256 byte，在此主要是为了供用户测试 I2C 总线而用，它并没有存储特定的参数。



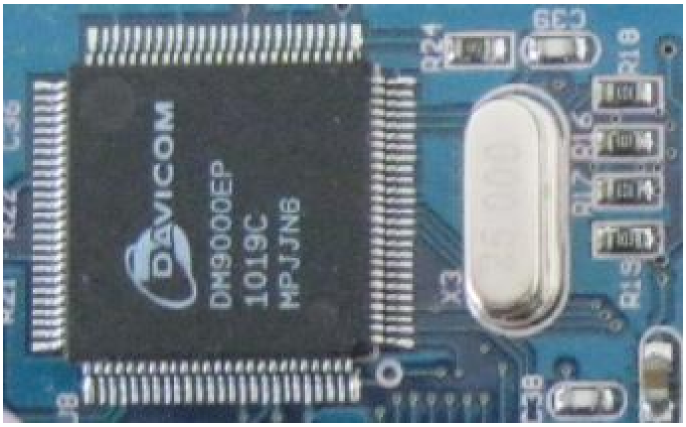
1.3.14 网络接口

本开发板采用了 DM9000 网卡芯片，它可以自适应 10/100M 网络，RJ45 连接头内部
已经包含了耦合线圈，因此不必另接网络变压器，使用普通的网线即可连接本开发板至你的路由器或者交换机。

注意：每个开发板的网络 MAC 地址都是相同的，它可以通过软件设定，对于 Linux 用户，本手册 2.4 章节有相关介绍；对于 WinCE 用户，您可以参考 BSP 里面的 DM9000 驱动代码和注册表文件(platform.reg)。



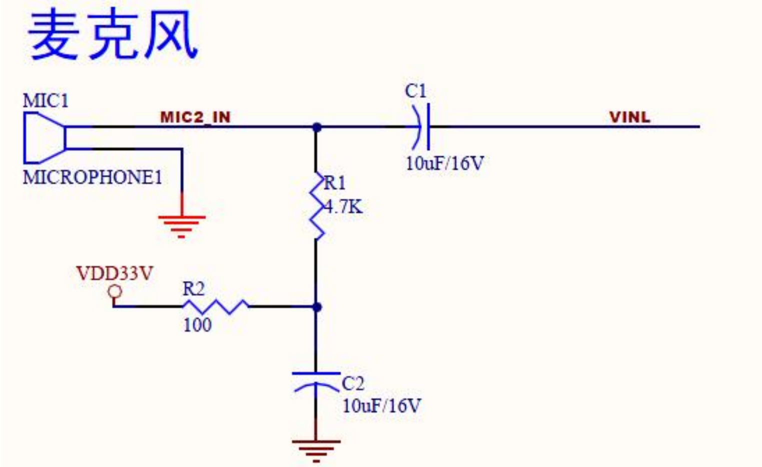




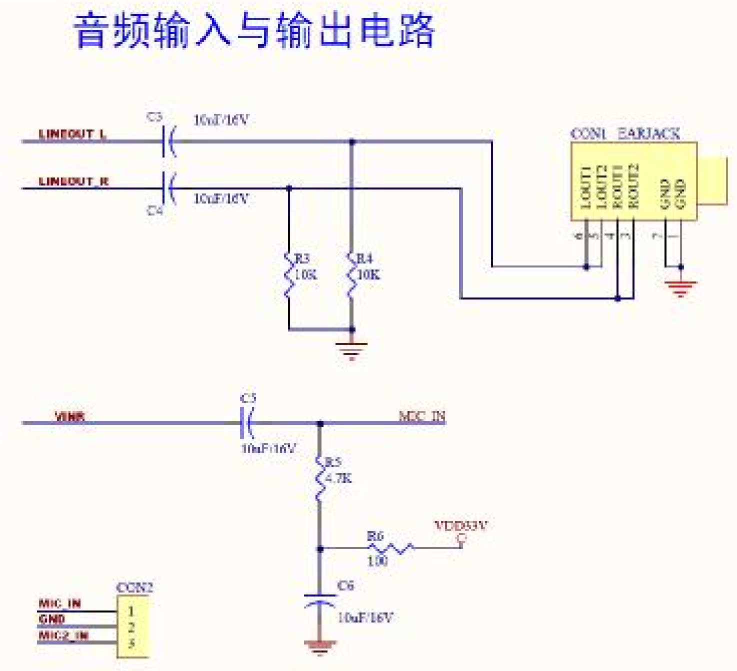
1.3.15 音频接口

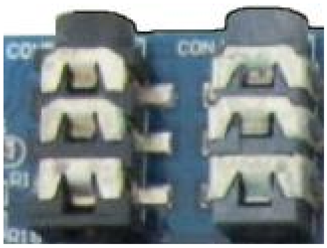
S3C2440 内置 I2S 总线接口，可直接外接 8/16 比特的立体声 CODEC，本开发板采用基于 I2S 总线的 UDA1341 芯片实现音频解码系统，该芯片内部寄存器的初始化和设置则是采用 L3-bus 总线连接控制实现的，在这里我们沿用了三星公板的设计，分别使用 CPU 的 GPB2、GPB3、GPB4 端口模拟实现 L3-Bus 规范的 L3MODE、L3DATA、L3CLOCK，它们在初始化完 UDA1341 以后就不再有用了，因此这三条控制线也可以使用普通的单片机模拟实现。

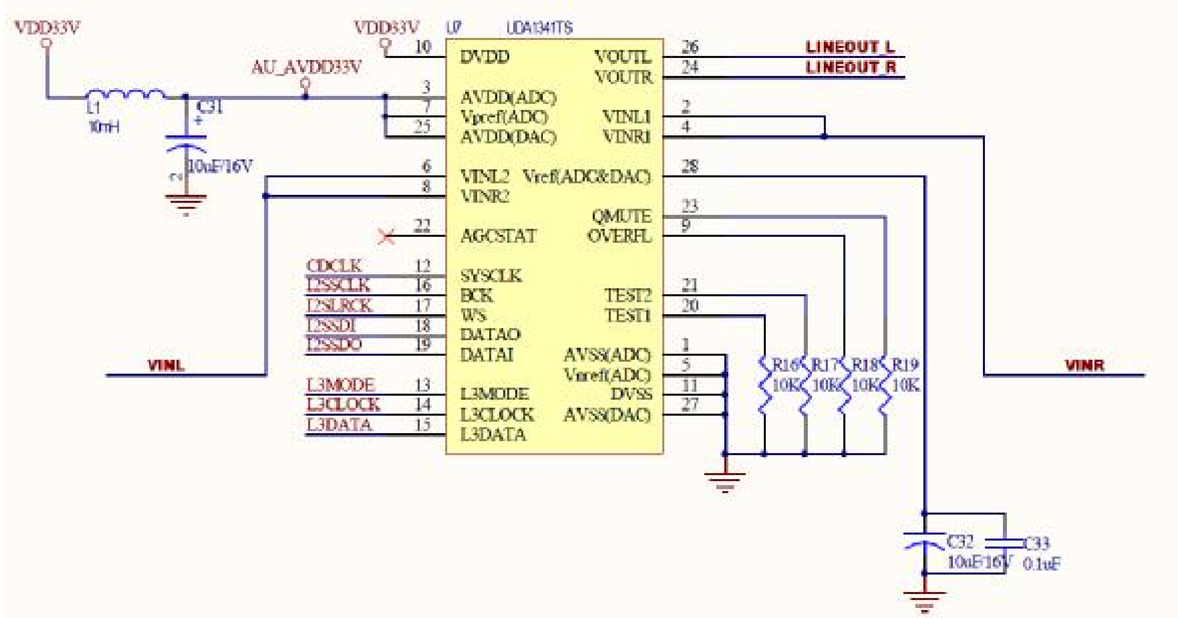
音频系统的输出为开发板上的常用 3.5mm 孔径插座，输入分为两路，一路为板载麦克风，另一路通过 CON2 白色 2.0mm 插座引出。两路音频输入通道的驱动是有些不同的，目前板载 MIC 对应的通道是可以录音使用的，注意：本开发板并非专业的录音设备，音频输入的处理电路很简单，录音时尽量把音源靠近麦克风。


麦克风
MIC1
MIC2_IN
C1
VINL
10uF/16V
MICROPHONE1
R1
4.7K
VDD33V
R2
100
C2
10uF/16V




音频输入与输出电路
LINEOUT_L
C3
10uF/16V
LINEOUT_R
C4
10uF/16V
CON1 EARJACK
LOUT1
LOUT2
ROUT1
ROUT2
GND
GND
R3
10K
R4
10K
VINR
C5
10uF/16V
MIC_IN
R5
4.7K
R6
100
VDD33V
C6
10uF/16V
CON2
MIC_IN
GND
MIC2_IN





1.3.16 JTAG 接口

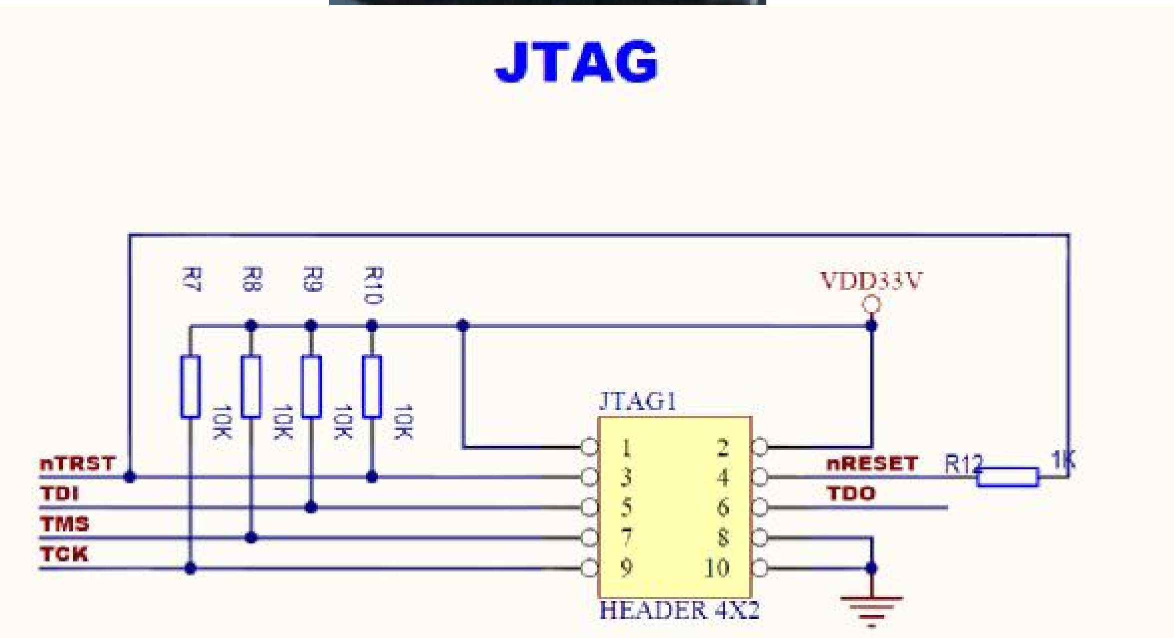
当开发板从贴片厂下线，里面是没有任何程序的，这时我们一般通过 JTAG 接口烧写第一个程序，就是 armvivi，借助 armvivi 可以使用 USB 口下载更加复杂的系统程序等，这在后面的章节中你可以看到。

除此之外，JTAG 接口在开发中最常见的用途是单步调试，不管是市面上常见的 JLINK 还是 ULINK，以及其他的仿真调试器，最终都是通过 JTAG 接口连接的。标准的 JTAG 接口是 4 线：TMS、 TCK、TDI、TDO，分别为模式选择、时钟、数据输入和数据输出线，加上电源和地，一般总共 6 条线就够了；为了方便调试，大部分仿真器还提供了一个复位信号。

因此，标准的 JTAG 接口是指是否具有上面所说的 JTAG 信号线，并不是 20Pin 或者 10Pin 等这些形式上的定义表现。这就如同 USB 接口，可以是方的，也可以扁的，还可以是其他形式的，只要这些接口中包含了完整的 JTAG 信号线，都可以称为标准的 JTAG 接口。本开发板提供了包含完整 JTAG 标准信号的 10 Pin JTAG 接口，各引脚定义如图。
说明：对于打算致力于Linux 或者WinCE 开发的初学者而言，JTAG 接口基本是没有任何意义和用途的，因为大部分开发板都已经提供了完善的BSP，这包括最常用的串口和网络

以及USB 通讯口，当系统装载了可以运行的Linux 或者WinCE 系统，用户完全可以通过这些高级操作系统本身所具备的功能进行各种调试，这时是不需要JTAG 接口的；即使你可以进行跟踪，但鉴于操作系统本身结构复杂，接口繁多，单步调试犹如大海捞针，毫无意义可言。想一想你手头使用的PC 机就知道了，或许你从没有见过甚至听过有谁会在PC主板上插一个仿真器，来调试PCI 这样接口的WindowsXP 或者Linux 驱动。这就是为什么你经常见到或者听到那么多人在讲驱动"移植"，因为大部分人都是参考前辈的实现来做驱动的。JTAG 仅对那些不打算采用操作系统，或者采用简易操作系统(例如uCos2 等)的用户有用。大部分开发板所提供的Bootloader 或者BIOS 已经是一个基本完好的系统了，因此也不需要单步调试。

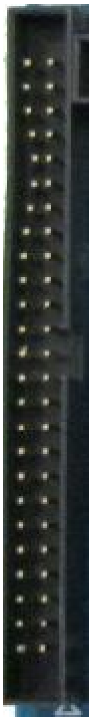




1.3.17 54pin 扩展接口

54pin扩展接口主要包括了常用GPIO和系统数据总线（LDATA0~LDATA15）的引出，

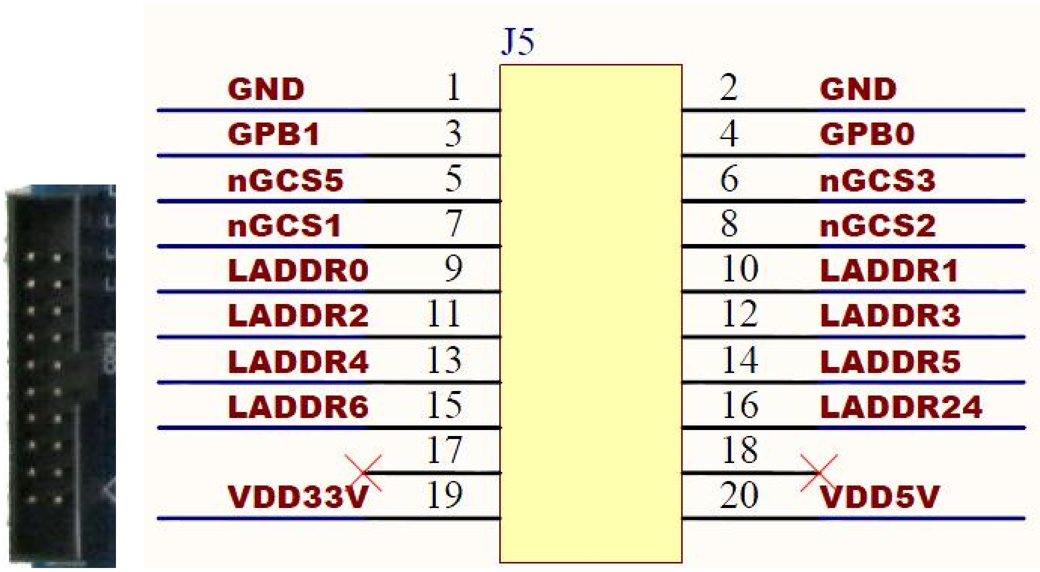
其中GPIO 是通用输入输出口的简称。如图，J4是一个54 Pin 2.0mm 间距的扩展接口。实际上，J4 不仅包含了很多富余的GPIO 引脚，还包含了一些其他CPU 引脚，如AD0-AIN3，CLKOUT 等。你所看到的图中的SPI 接口、I2C 接口、GPB0 和GPB1 等，它们其实也是GPIO，不过是以特殊功能接口来标称定义的，这些都可以通过相应的CPU 寄存器来设置更改它们的用途，详细的接口用途请查阅S3C2440数据手册。





1.3.18 20pin 扩展接口

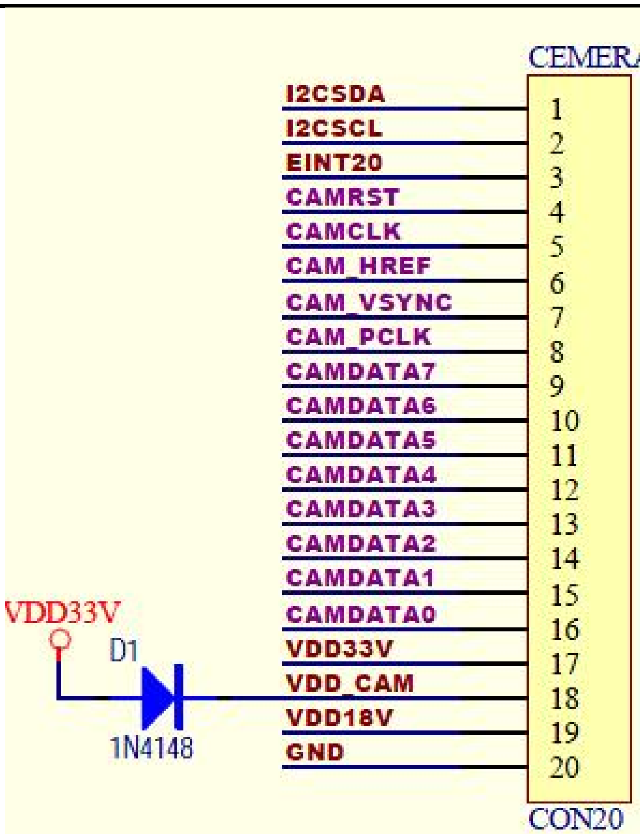
本开发板上还有一个20pin的扩展接口J5，它主要是将8 条地址(A0-A6, A24)和一些控制信号线(片选、读写等)引出，J5 可以向外提供5V 电压输出；实际上，很少有用户通过总线扩展外设。



1.3.19 CMOS CAMERA 接口

S3C2440 带有CMOS 摄像头接口，在开发板上通过标称为CAMERA 的接口引出。它是一个20 脚2.0mm 间距的针座，用户可以直接使用我们提供的CAM130 摄像头模块；其实CAM130 摄像头模块上面没有任何电路，它只是一个转接板，它直接连接使用了型号为ZT130G2 摄像头模块，它们的定义如下图所示。说明：CAMERA 接口是一个复用端口，它可以通过设置相应的寄存器改为GPIO 使用，下表是它对应引脚的GPIO 列表。





CAMERA	网络名称	可复用为	CAMERA	网络名称	可复用为
1	I2CSDA	GPE15	2	I2CSCL	GPE14
3	EINT20	GPG12	4	CAMRST	GPJ12
5	CAMCLK	GPJ11	6	CAM_HREF	GPJ10
7	CAM_VSYNC	GPJ9	8	CAM_PCLK	GPJ8
9	CAMDATA7	GPJ7	10	CAMDATA6	GPJ6
11	CAMDATA5	GPJ5	12	CAMDATA4	GPJ4
13	CAMDATA3	GPJ3	14	CAMDATA2	GPJ2
15	CAMDATA1	GPJ1	16	CAMDATA0	GPJ0

17	VDD33V	3.3V	电源18	VDD_CAM	VDD_CAM
19	VDD18V	1.8V	电源20	GND	地

第2章 GQ2440 开发板使用说明

出厂之前，如果客户未加说明，我们一般已经烧写缺省的 linux 系统(包含三个文件，对应的光盘二进制文件是 armvivi、zImage_t35、root_qtopia.img)，请注意以下的操作是基于 WindowsXP 环境的。

2.1 开发板设置及连接

2.1.1 启动模式选择





本开发板的启动模式选择，是通过按下开发板上对应按键来决定的。

根据目标板提示:

POW_KEY: 按下系统将从 Nand Flash 启动;

Nor_KEY: 按下系统将从 Nor Flash 启动。

出厂的时候开发板的 Nor Flash 和 Nand Flash 已经烧入了相同的 BIOS(因为该 BIOS 同时支持这两种 Flash，只是开机后表现形式不同，请参考"开发板 BIOS 功能及使用说明"一节)。两个按键的具体操作情况如下表:

操作前状态	操作	操作后状态
未上电	按下 POW_KEY	从 Nand Flash 启动
未上电	按下 Nor_KEY	从 Nor Flash 启动
正从 Nor 启动	按下 POW_KEY	关机
	按下 Nor_KEY	关机
正从 Nand 启动	按下 POW_KEY	关机
	按下 Nor_KEY	从 Nor Flash 启动

2.1.2 外部接口连接

- 请使用我们提供的直连串口线连接开发板的串口和 PC 机的串口

- 用我们提供的交叉网线将开发板的网络接口与 PC 相连
- 用我们提供的 5V 电源适配器连接到板上的 5V 输入插座
- 把音箱或者耳机的插头接入板上的音频输出口(靠右)
- 如果您有液晶屏，请按照数据线头的方向与开发板的 LCD 接口相连
- 用 USB 电缆连接开发板和 PC

2.1.3 设置超级终端

为了通过串口连接开发板，必须使用一个模拟终端程序，几乎所有的类似软件都可以使用，其中 MS-Windows 自带的超级终端是最常用的选择，当你安装 Windows9x 时需要自定义选择安装该项，Windows2000 及更高版本则已经缺省安装。一般桌面版 Linux 系统也自带了类似的串口终端软件，叫 minicom，它是基于命令行的程序，使用比较复杂一些，感兴趣的用户可以在网上找一下这方面的介绍。在此着重介绍一下 Windows 自带的超级终端程序并以 WindowsXP 为例，或许其他 Windows 版本的程序界面有所不同。

通过【开始】|【所有程序】|【附件】|【通讯】|【超级终端】，打开超级终端，按照如下图所示的步骤创建一个适合的串口连接配置。

超级终端会要求你为新的连接取一个名字，如图所示，这里我取了”com”，Windows 系统会禁止你取类似”COM1”这样的名字，因为这个名字被系统占用了。



当你命名完以后，又会跳出一个对话框，你需要选择连接开发板的串口，我这里选择了串口 3，如图所示。



最后，**最重要的一步是设置串口，注意必须选择无流控制，否则，或许你只能看到输出而不能输入，另外板子工作时的串口波特率是 115200**，如图所示。



点击【确定】后就配置好了一个超级终端的串口连接了。

当所有的连接参数都设置好以后，打开电源开关，系统会出现 vivi 启动界面。选择超级终端“文件”菜单下的“另存为…”，保存该连接设置，以便于以后再连接时就不必重新执行以上设置了。

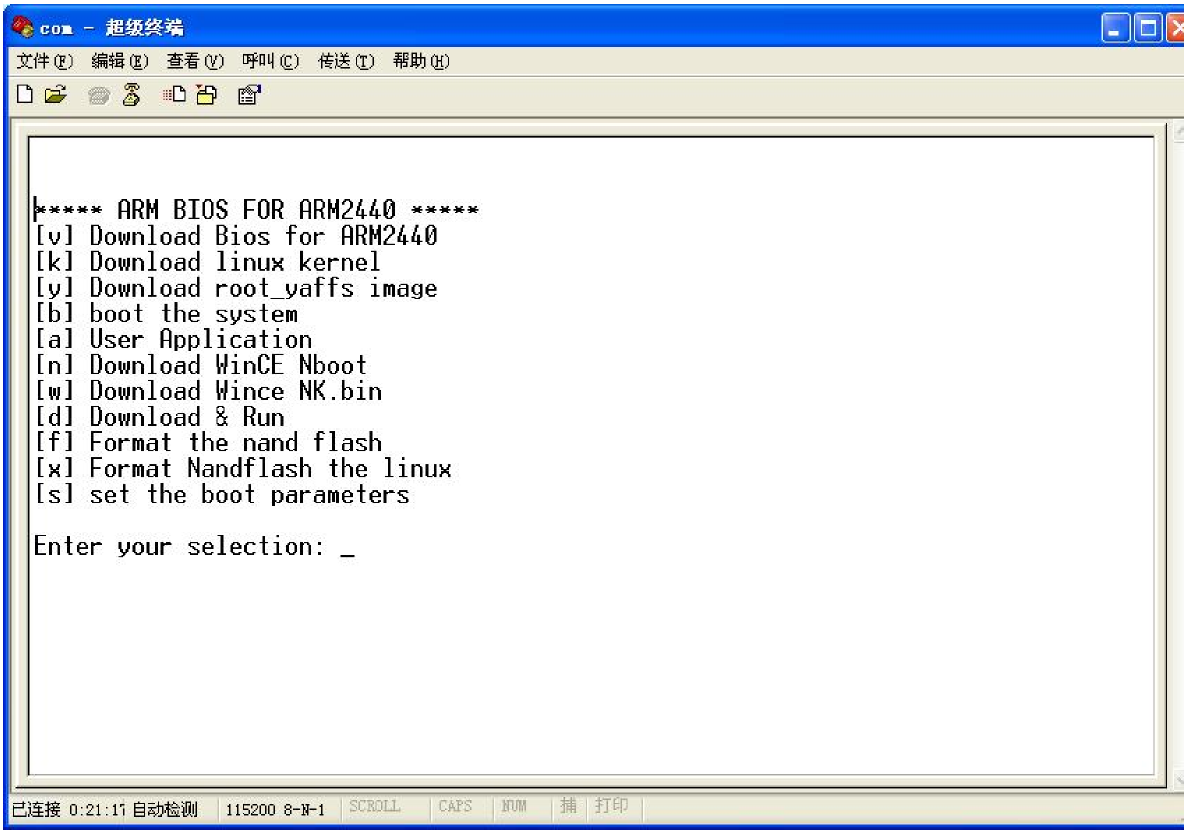
windows 7 上没有超级中断，可以使用其他串口软件替代，例如 SecureCRT。

2.2 开发板 BIOS 功能及使用说明

2.2.1 开机进入 BIOS 模式



armvivi 在出厂的时候已经预装入板子的 Nor Flash 中，按下开关 Nor_KEY 从 Nor Flash 启动，即可进入 BIOS 模式，此时开发板上 Nor_KEY 旁边的蓝色 NOR_LED1 会点亮，4 个用户 LED 中的 LED1 会呈现闪烁状态，其启动界面如下图：



armvivi 简介:

开发板采用的 BIOS 是基于三星原来的 bootloader 之 vivi 改进而来，名为 armvivi，它采用功能菜单的方式，并可以和原来的命令交互模式互相切换。

armvivi 可以使用 JTAG 板(一般借助 H-JTAG 软件)直接烧写入 Nor Flash 中使用，也可以直接烧入 Nand Flash 中运行。当烧入 Nor Flash 并从中时，将会出现菜单模式；当烧入 Nand Flash 并从中运行时，按下开发板上的任意一个按键，也可以出现菜单模式，否则会启动开发板上预装的操作系统(Linux/WindowsCE)。

armvivi 的菜单模式主要为烧写系统和调试而用，也可以设置参数和进行分区等，它采用 USB 下载的方式，因此搭建烧写环境极为简单，并且下载速度快，使用十分方便。

如果 armvivi 被烧写入 Nor Flash(默认)，您不仅可以用它来方便的下载更新 linux 和 WinCE 系统，还可以烧写其他任何支持 Nand Flash 启动的操作系统和非操作系统到 NandFlash，如 uCos2，U-boot，Nboot，2440test 等，然后再选择系统从 Nand Flash 启动，这样您就可以使用各种各样的系统了，我们将会逐步增加这方面的 Demo 文件，请留意我们的网站信息。

如果 armvivi 被烧写入 Nand Flash，它可以自动识别您烧写的 Linux 或者 WindowsCE 系统、或者其他系统，并快速自动启动它们。在本手册的“安装和更新系统”一节，我们就直接使用它来作为 bootloader。

另外，使用 Download & Run 功能，您还可以把程序下载到内存马上运行，这对于开发调试是极有帮助的，这样，您甚至不使用仿真器都可以了，我们光盘中的 2440test 程序就是这样一个例子。

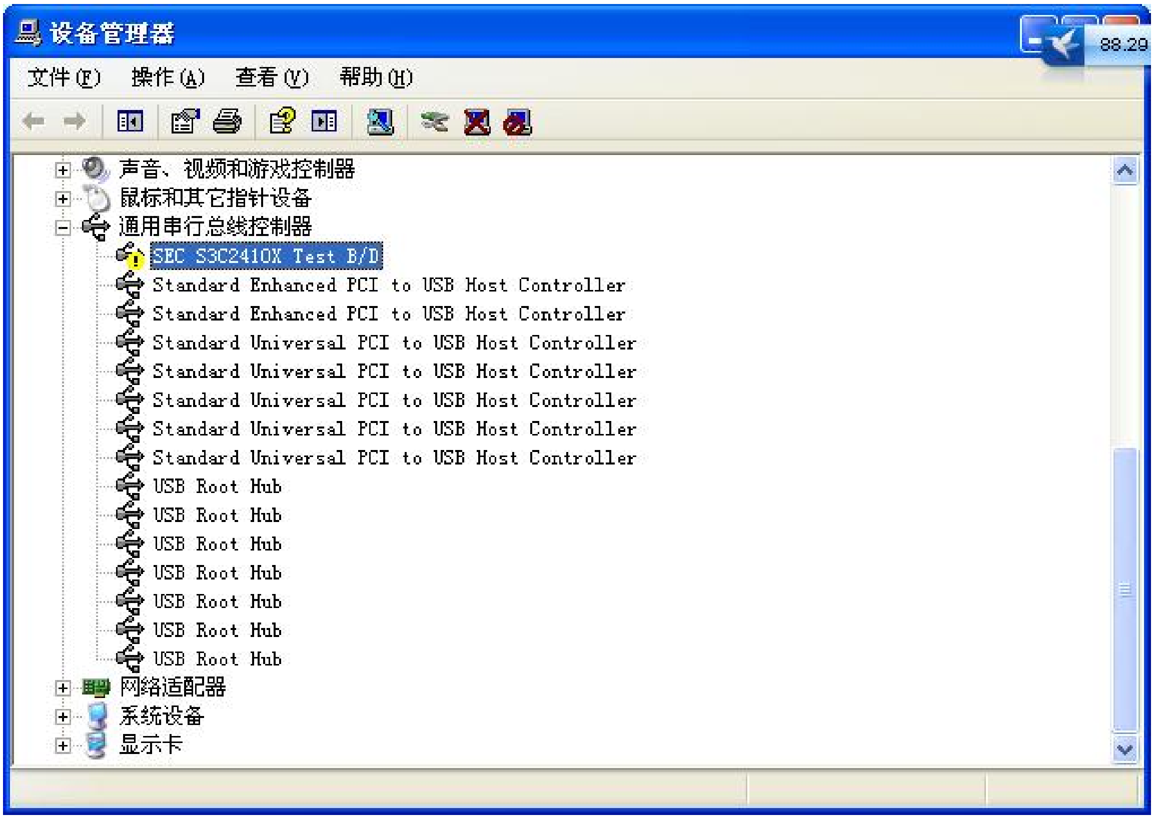
使用 armvivi 还可以把 Linux 内核文件 zImage 直接下载到内存中运行，如果您在 armvivi 中设定好网络启动参数，则还可以通过网络启动整个系统；同样的，armvivi 也可以把 WinCE 的运行时映像文件 NK.nb0 下载在内存中运行。

2.2.2 安装 USB 下载驱动

注意：此处安装的 USB 驱动仅在 BIOS 模式下有用，它需要配合 dnw.exe 软件使用，进入 Linux 或者 WinCE 系统都不会使用到该驱动。

第一次用 USB 下载线将开放板连接到 PC 上时 windows 会提示发现新硬件并要求安装驱动。我们可以进入 windows 的【设备管理器】，在【通用串行总线控制器】一项中发现这

个设备，如图下图所示。



注意：只有在运行 supervivi 时才会提示发现新硬件。

双击这个带有感叹号的设备进入设备属性菜单，如图下图所示，点击 重新安装驱动程序(I)... 按钮。

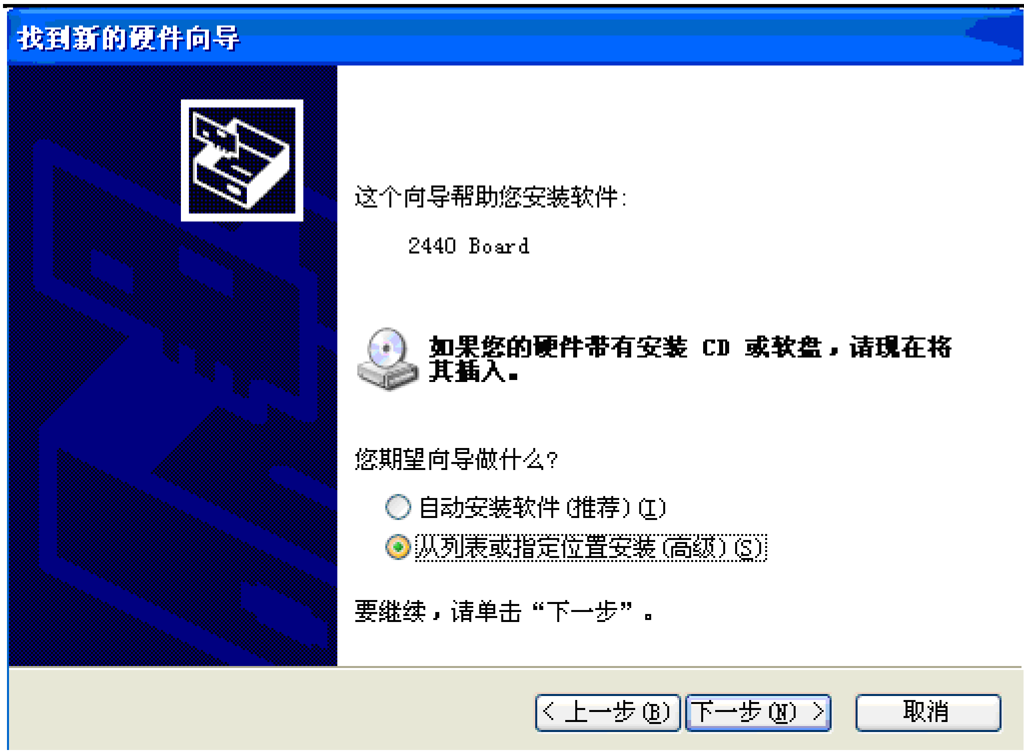


图 3-1　　设备属性

经过上一步会进入到“找到新硬件向导”，请按照如下图所示的步骤安装此设备的驱动。

选择不连接到 Windows Updata 升级:


找到新的硬件向导
欢迎使用找到新硬件向导
Windows 将通过在计算机、硬件安装 CD 或 Windows Update 网站(在您允许的情况下)上查找来搜索当前和更新的软件。
阅读隐私策略
Windows 可以连接到 Windows Update 以搜索软件吗?
是，仅这一次(Y)
是，这一次和每次连接设备时(E)
否，暂时不(T)
单击"下一步"继续。
< 上一步(B)
下一步(N) >
取消



选择从列表或指定位置安装:

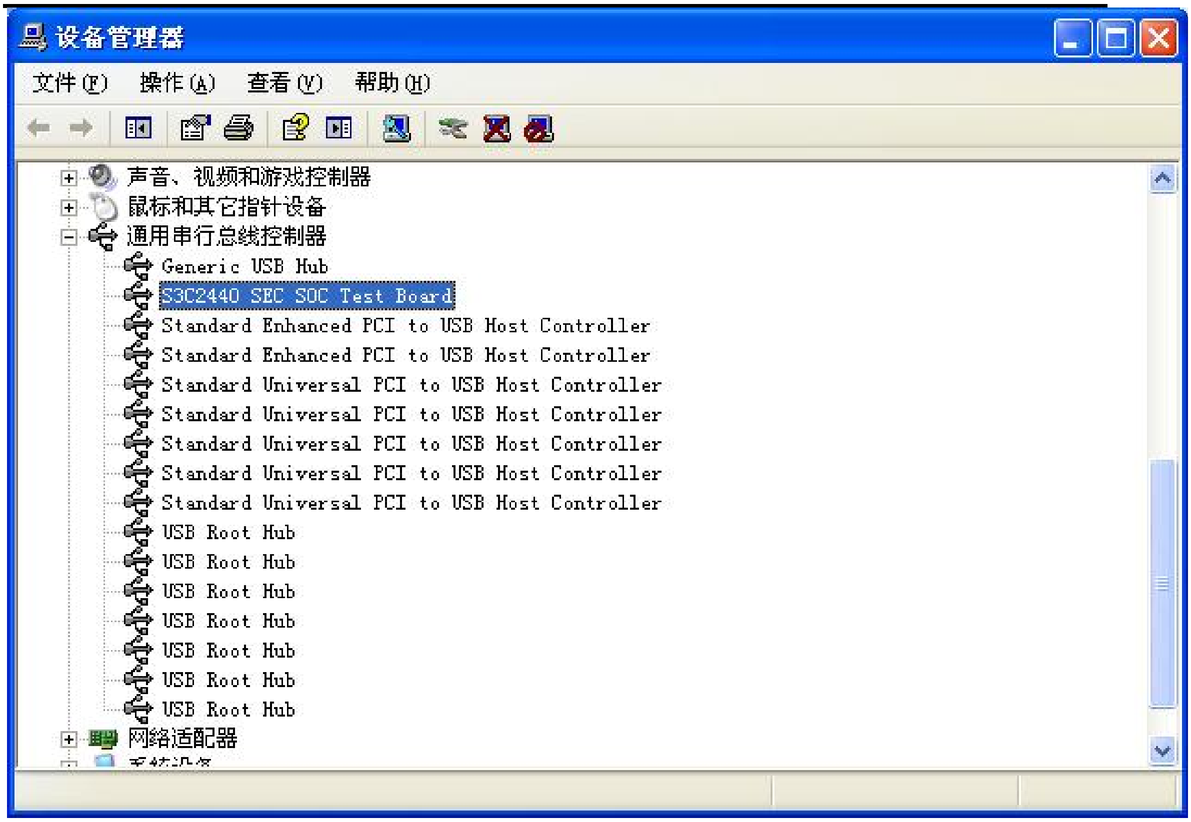
选择指定的搜索目录，这里我们设置为光盘中存放该驱动的目录“windows 平台工具\usb 下载驱动\”。



从搜索到的驱动中选择与自己的开发板匹配的驱动，然后点击【下一步】就可以进行安装了。



在完成安装后，我们可以通过【设备管理器】中看到 PC 检测到指定设备的信息，另外我们还可以通过 USB 下载软件 dnw.exe 来验证 USB 下载驱动的成功安装，看到下图 DNW 的标题栏上的【USB:OK】就说明 USB 下载驱动已经安装成功了。

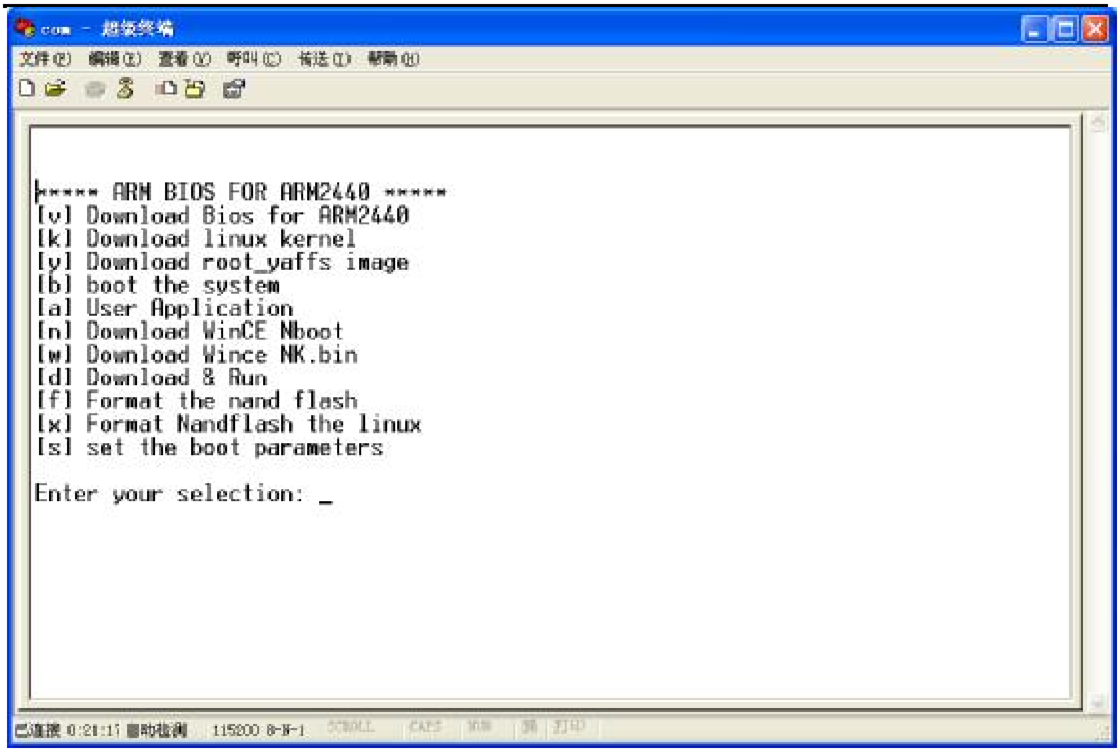


设备管理器
文件(F) 操作(A) 查看(V) 帮助(H)
声音、视频和游戏控制器
鼠标和其它指针设备
通用串行总线控制器
Generic USB Hub
S3C2440 SEC SOC Test Board
Standard Enhanced PCI to USB Host Controller
Standard Enhanced PCI to USB Host Controller
Standard Universal PCI to USB Host Controller
Standard Universal PCI to USB Host Controller
Standard Universal PCI to USB Host Controller
Standard Universal PCI to USB Host Controller
Standard Universal PCI to USB Host Controller
USB Root Hub
USB Root Hub
USB Root Hub
USB Root Hub
USB Root Hub
USB Root Hub
USB Root Hub
网络适配器



2.2.3 功能主菜单说明

注意：以下通过 USB 下载的功能均配合 DNW 这个程序使用。



功能[v]: 通过 USB 下载 Linux bootloader 到 Nand Flash 的 bootloader 分区。

功能[k]: 通过 USB 下载 Linux 内核到 Nand Flash 的 kernel 分区。

功能[y]: 通过 USB 下载 yaffs 文件系统映象到 Nand Flash 的 root 分区。

功能[b]: 启动系统，如果烧入了 linux 或者 wince，执行从命令将自动辨认识别启动系统。

功能[a]: 通过 USB 下载用户程序到 Nand Flash 中，一般这样的用户程序为 bin 可执行文件，如 2440test(需要支持超过 4K 限制)、uCos2(开发板中带的 uCos2 支持 nand flash 启动)、

U-Boot 等；当然也可以是其他任意大小的 bin 程序。

功能[n]: 通过 USB 下载 WinCE 之启动程序 Nboot 到 Nand Flash 的 Block0。

功能[w]: 通过 USB 下载 WinCE 发行映象 NK.bin 到 Nand Flash。

功能[d]: 通过 USB 下载程序到指定内存地址(通过 DNW 的 Configuration->Option 选项指定运行地址)，并运行。对于本开发板，SDRAM 的物理起始地址是 0x30000000，结束地址

是 0x34000000，大小为 64Mbytes，另外 BIOS 本身占用了 0x33DE8000 以上的空间，

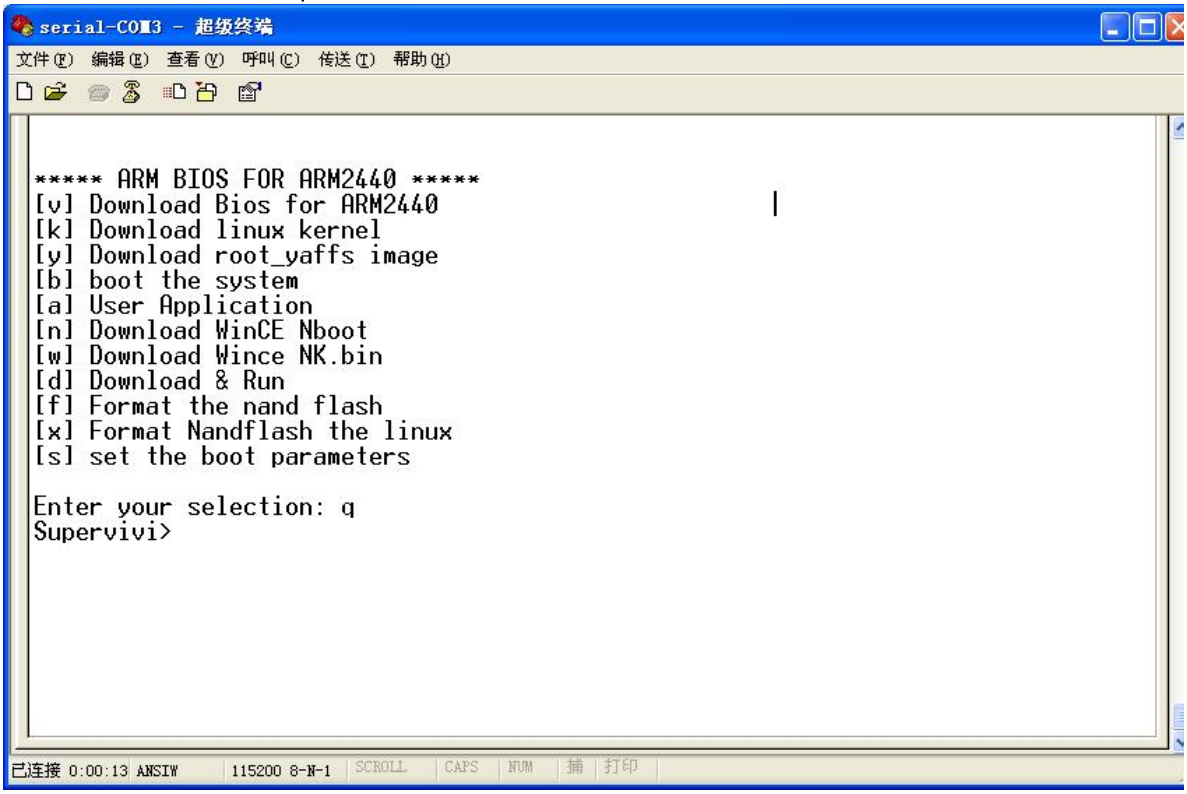
因此在用 BIOS 的 USB 下载功能时应指定地址在 0x30000000 - 0x33DE8000 之间。

功能[f]: 擦除 Nand Flash，执行此功能将会擦除整片 Nand Flash 中的数据。(如果您是第一次使用本开发板，请不必担心误操作，您可以根据本手册第三章的步骤恢复到出厂状态)

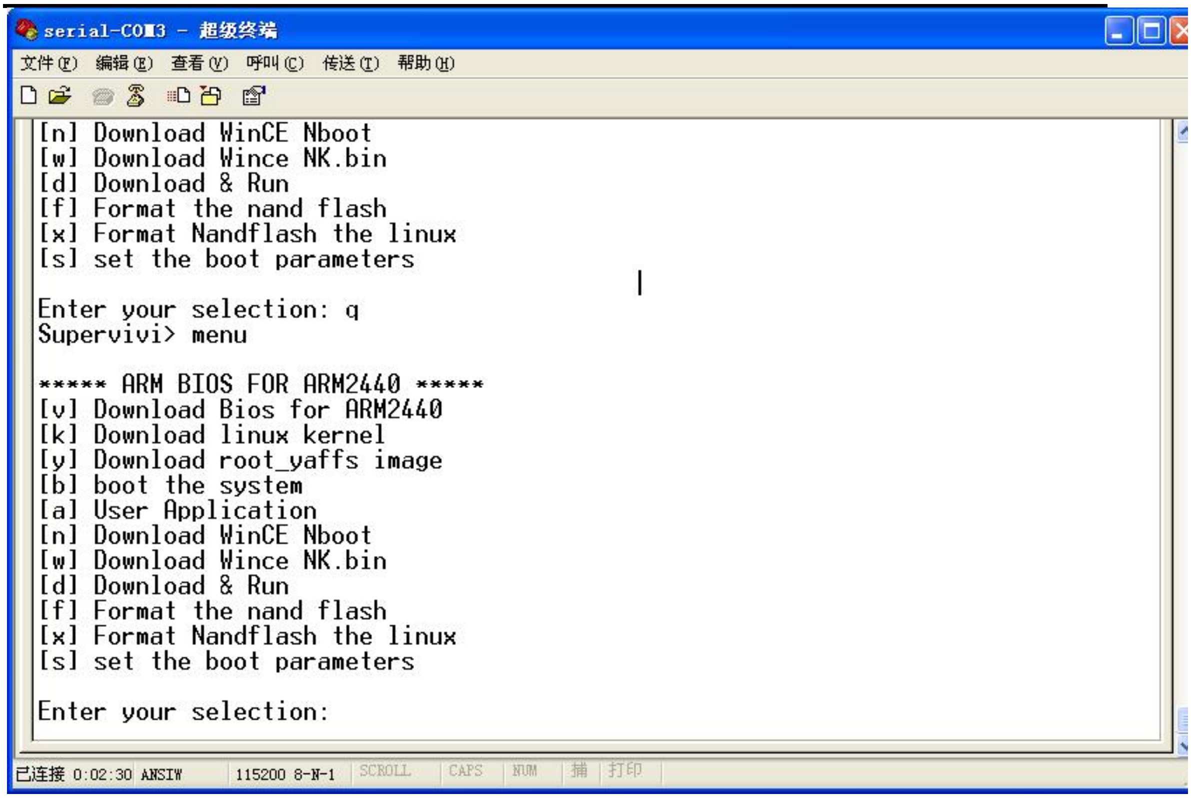
功能[x]: 对 Nand Flash 进行默认分区,相当于执行命令行的 bon part 0 320k 2368k，此命令仅对 Linux 系统有效。

功能[s]: 设置 linux 启动参数，详细见子菜单说明。

在菜单模式下，按 q 可以进入命令行模式:

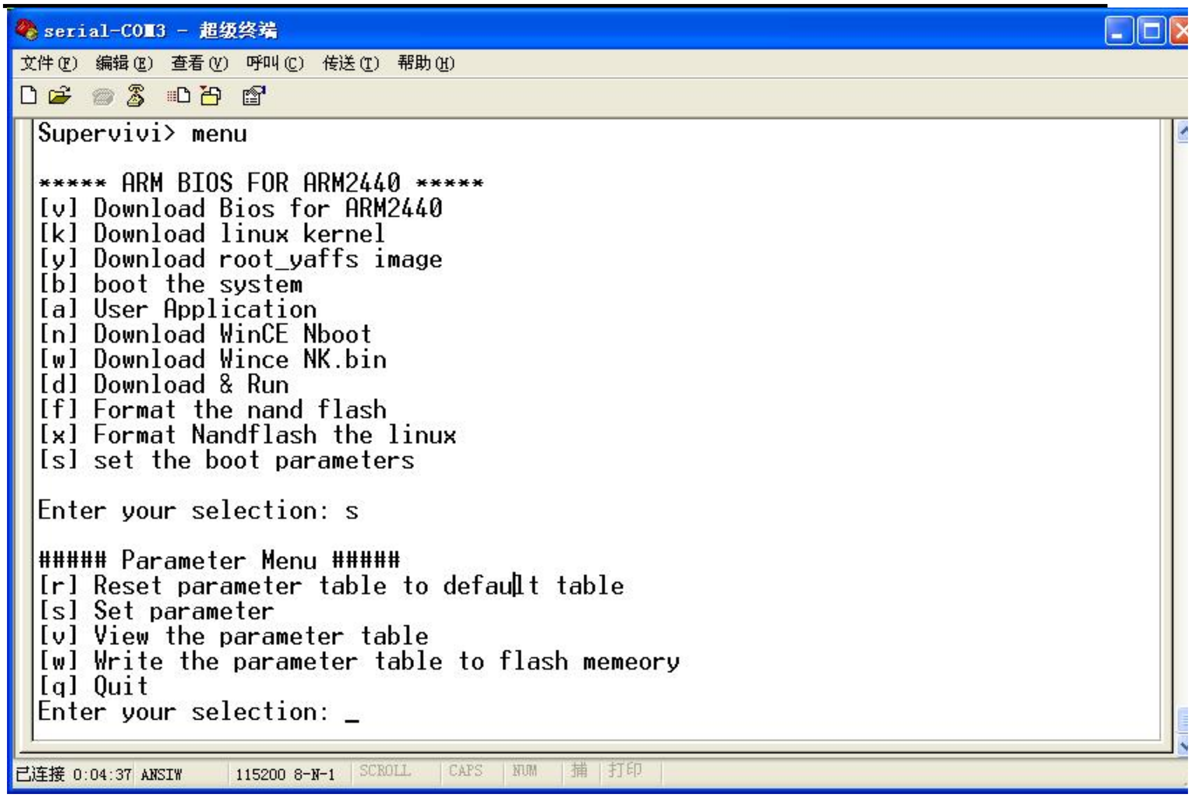


在命令行模式下输入 menu 命令可以返回菜单模式。



2.2.4 设置 Linux 启动参数子菜单功能说明

通过该子菜单功能，可以更加灵活的启动 Linux 系统，在 BIOS 主菜单执行功能号[s]，进入设置 Linux 启动参数子菜单，如图：




```
Supervivi> menu

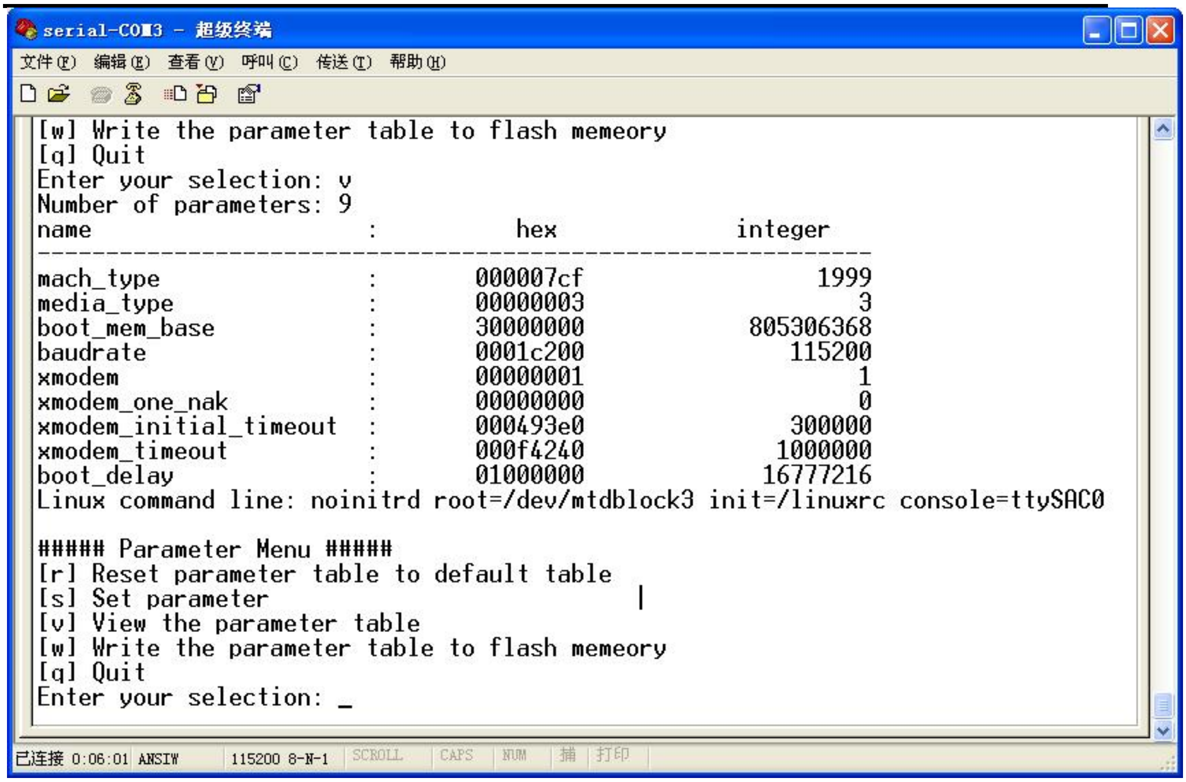
***** ARM BIOS FOR ARM2440 *****
[v] Download Bios for ARM2440
[k] Download linux kernel
[y] Download root_yaffs image
[b] boot the system
[a] User Application
[n] Download WinCE Nboot
[w] Download Wince NK.bin
[d] Download & Run
[f] Format the nand flash
[x] Format Nandflash the linux
[s] set the boot parameters

Enter your selection: s

##### Parameter Menu #####
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: _
```

(1)浏览当前参数设置[v]

输入"v"可以浏览当前启动参数设置情况:



```
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: v
Number of parameters: 9
name                    :         hex              integer
-------------------------------------------------------------
mach_type               :      000007cf                1999
media_type              :      00000003                   3
boot_mem_base           :      30000000           805306368
baudrate                :      0001c200              115200
xmodem                  :      00000001                   1
xmodem_one_nak          :      00000000                   0
xmodem_initial_timeout  :      000493e0              300000
xmodem_timeout          :      000f4240             1000000
boot_delay              :      01000000            16777216
Linux command line: noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0

##### Parameter Menu #####
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: _
```

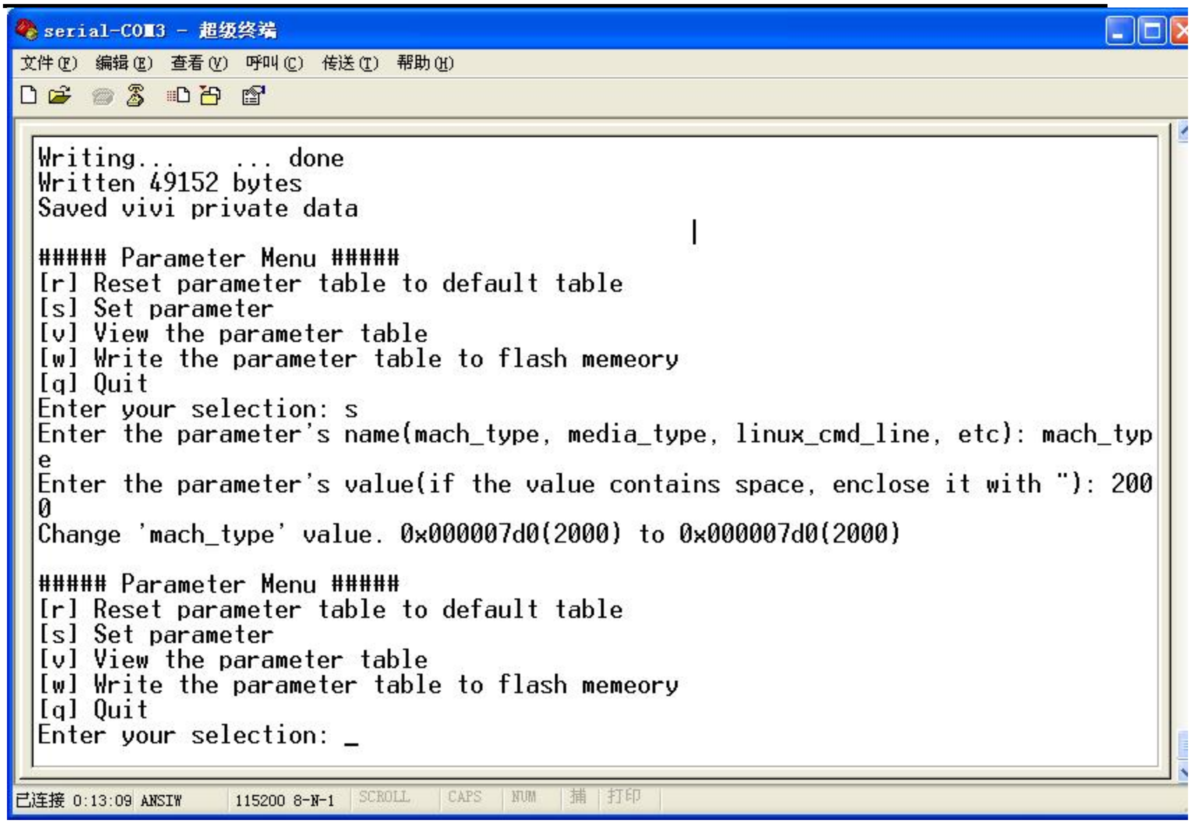
(2)设置参数[s]

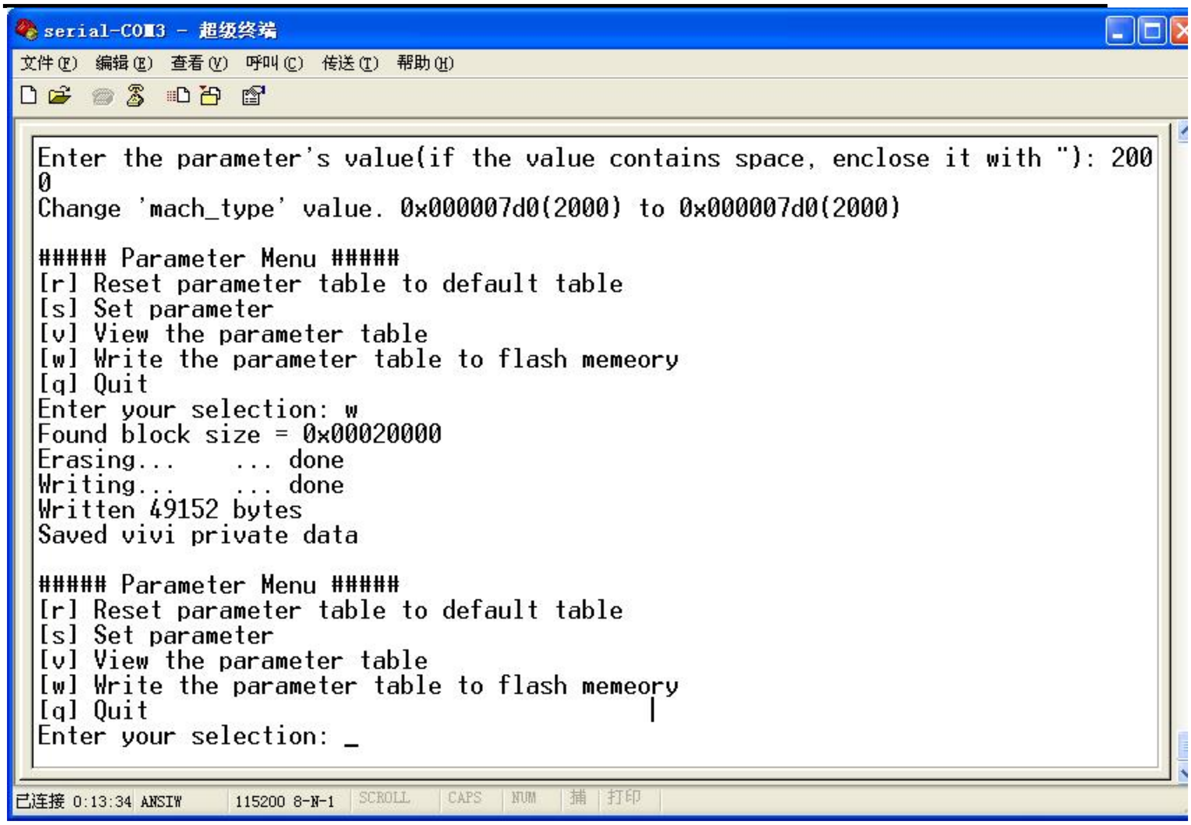
输入“s”，可以对上面列出的参数进行设置，比较常用的参数有(其他参数建议不要更改):

- Mach_type
- Linux command line

下面我们分别举例说明如何设置:

开发板默认的 MACH_TYPE 为 1999，假设你编译的内核使用的 MACH_TYPE 是 2000，则可以通过修改 mach_type 参数来正常启动内核，根据提示先输入参数的名字“mach_type”，再输入参数值“2000”(引号不要输入)，更改后记得输入“w”保存设置，如图:


serial-COM3 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Writing... ... done
Written 49152 bytes
Saved vivi private data
Parameter Menu
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: s
Enter the parameter's name(mach_type, media_type, linux_cmd_line, etc): mach_type
Enter the parameter's value(if the value contains space, enclose it with "): 2000
Change 'mach_type' value. 0x000007d0(2000) to 0x000007d0(2000)
Parameter Menu
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: _
已连接 0:13:09 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印



Linux_cmd_line 是经常用到的一个内核启动参数，例如要把内核的启动信息和登录终端改为串口 1（默认是串口 0），则这样修改：

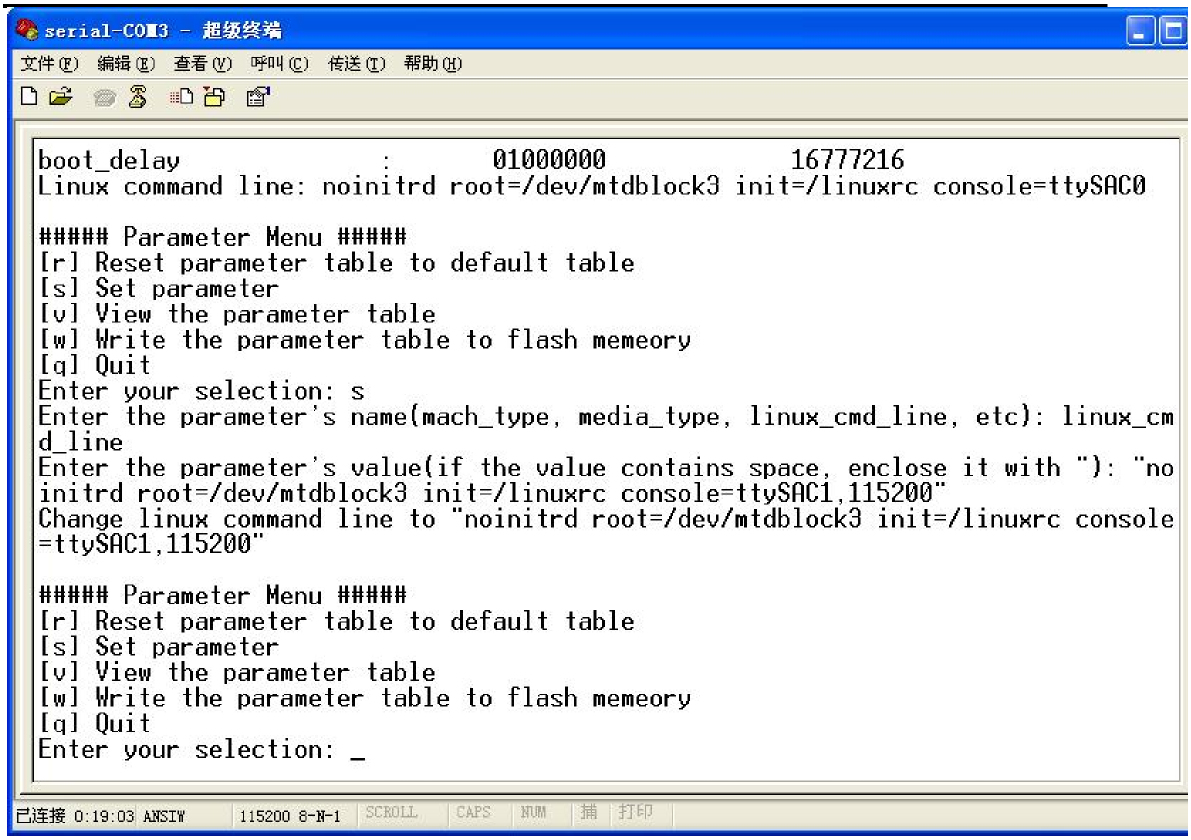
通过浏览参数，可以看到原来的参数：

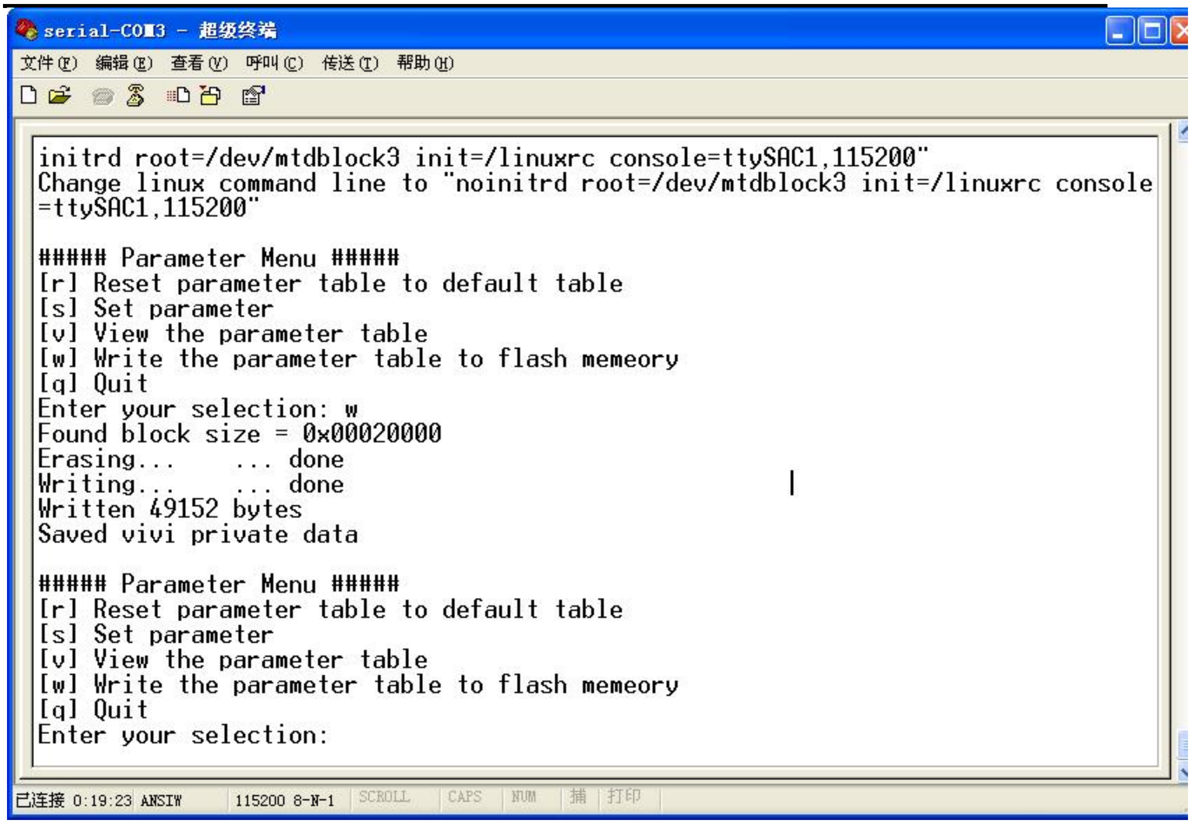
Linux_cmd_line：noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0

输入“s”后，根据提示输入要修改的参数“linux_cmd_line”，回车，再输入参数值为(因为该参数串中有空格，因此需要输入双引号括起来)：

“noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC1,115200”

如图所示：


serial-COM3 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
boot_delay : 01000000 16777216
Linux command line: noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0
Parameter Menu
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: s
Enter the parameter's name(mach_type, media_type, linux_cmd_line, etc): linux_cm
d_line
Enter the parameter's value(if the value contains space, enclose it with "): "no
initrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC1,115200"
Change linux command line to "noinitrd root=/dev/mtdblock3 init=/linuxrc console
=ttySAC1,115200"
Parameter Menu
[r] Reset parameter table to default table
[s] Set parameter
[v] View the parameter table
[w] Write the parameter table to flash memeory
[q] Quit
Enter your selection: _
已连接 0:19:03 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印



这样系统启动的时候，内核的启动信息和登录信息都将在串口 1 出现，而 vivi 的输出信息不会改变，还是从串口 0 出来。

(3)保存配置[w]

当设置更改之后，可以输入“w”保存所作的更改。

(4)恢复默认值[r]

输入“r”可以恢复出厂时的内核启动参数。

(5)返回主菜单[q]

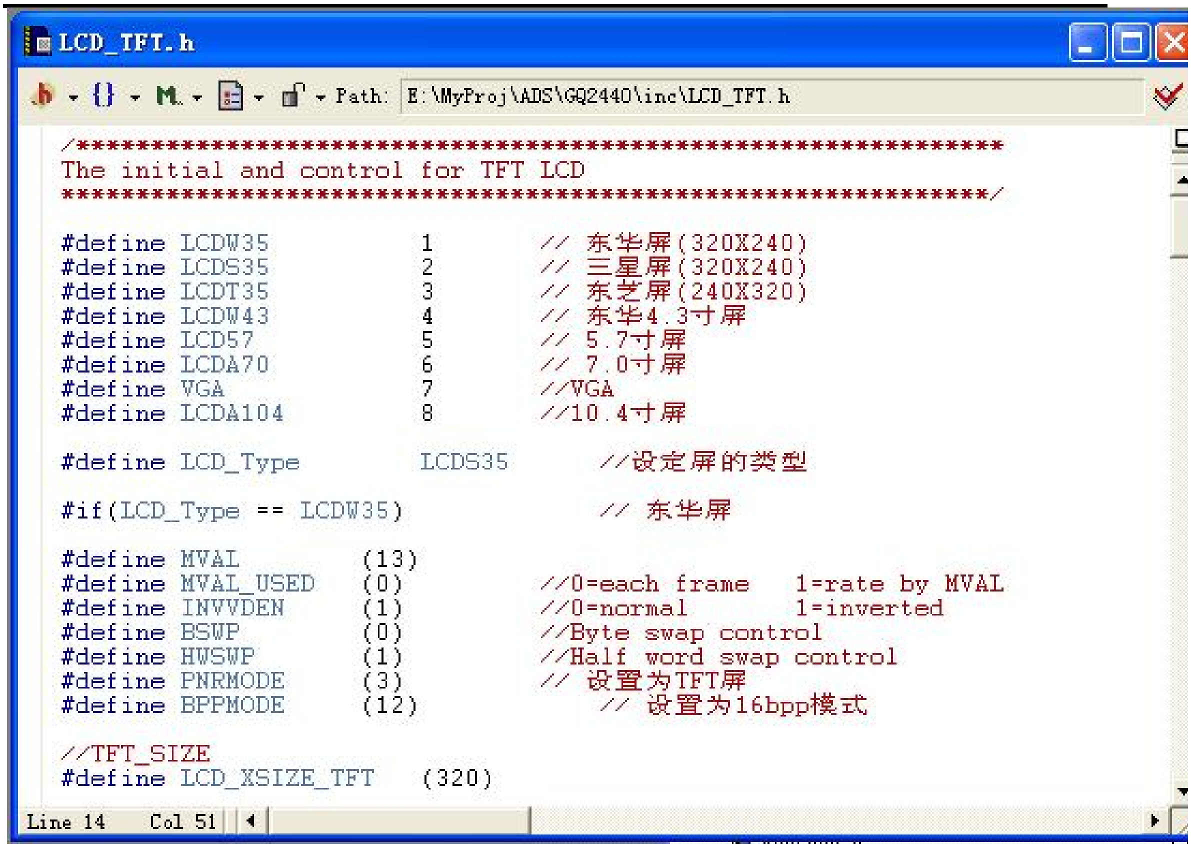
输入“q”可以返回 BIOS 功能主菜单。

2.3 非操作系统下的外围资源测试

在非操作系统下，主要测试 PWM 控制蜂鸣器，RTC 实时时钟测试，AD 转换测试，按键，触摸屏，各种 LCD，红外测试，I2C 总线测试，音频输入与输出，SD 卡功能。

2.3.1 下载运行测试程序

说明：2440test 是一个裸机测试程序，它不是一个操作系统，该程序由三星原厂的同名文件修改而来，我们根据实际情况，更改了输出菜单和各项测试内容，使其更加简洁明了，用户使用本节的步骤通过 USB 下载到内存中即可运行。我们提供的 2440test.bin 镜像文件是针对标配三星 3.5 屏的，如果使用其他类型的屏幕，只需在头文件中更改一下 LCD 类型(2440test\inc\LCD_TFT.h 中“LCD_TYPE”定义)后重新编译成 bin 文件即可。如下图所示。



LCD_TFT.h

Path: E:\MyProj\ADS\GQ2440\inc\LCD_TFT.h

```
/*****************************************************************
The initial and control for TFT LCD
*****************************************************************/

#define LCDW35          1       // 东华屏(320X240)
#define LCDS35          2       // 三星屏(320X240)
#define LCDT35          3       // 东芝屏(240X320)
#define LCDW43          4       // 东华4.3寸屏
#define LCD57           5       // 5.7寸屏
#define LCDA70          6       // 7.0寸屏
#define VGA             7       //VGA
#define LCDA104         8       //10.4寸屏

#define LCD_Type        LCDS35      //设定屏的类型

#if(LCD_Type == LCDW35)             // 东华屏

#define MVAL            (13)
#define MVAL_USED       (0)     //0=each frame   1=rate by MVAL
#define INVVDEN         (1)     //0=normal       1=inverted
#define BSWP            (0)     //Byte swap control
#define HWSWP           (1)     //Half word swap control
#define PNRMODE         (3)     // 设置为TFT屏
#define BPPMODE         (12)        // 设置为16bpp模式

//TFT_SIZE
#define LCD_XSIZE_TFT   (320)
```

Line 14 Col 51

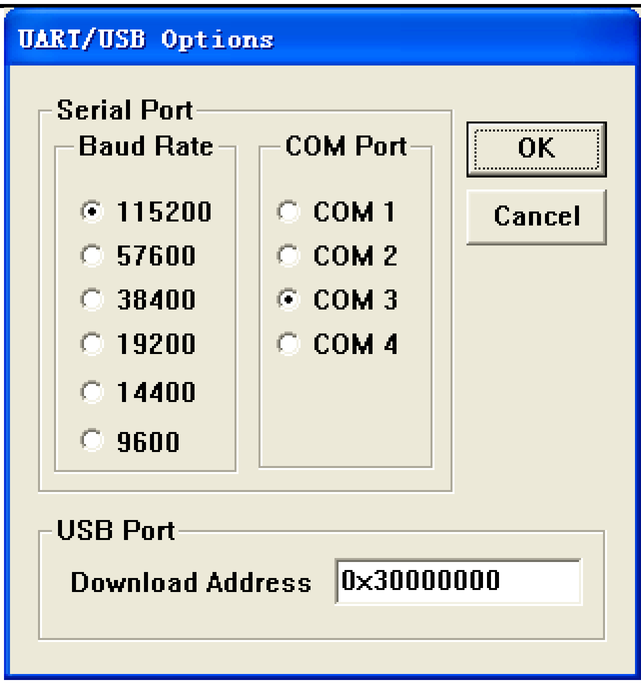
在光盘“images\2440test\”目录中找到 2440test.bin 文件,通过 BIOS 下载运行该测试程序，步骤如下:

(1)连接好开发板电源，串口线，USB 线，并通过按键开关 Nor_KEY 从 Nor Flash 启动系统，分别打开串口超级终端和 DNW，上电启动开发板。

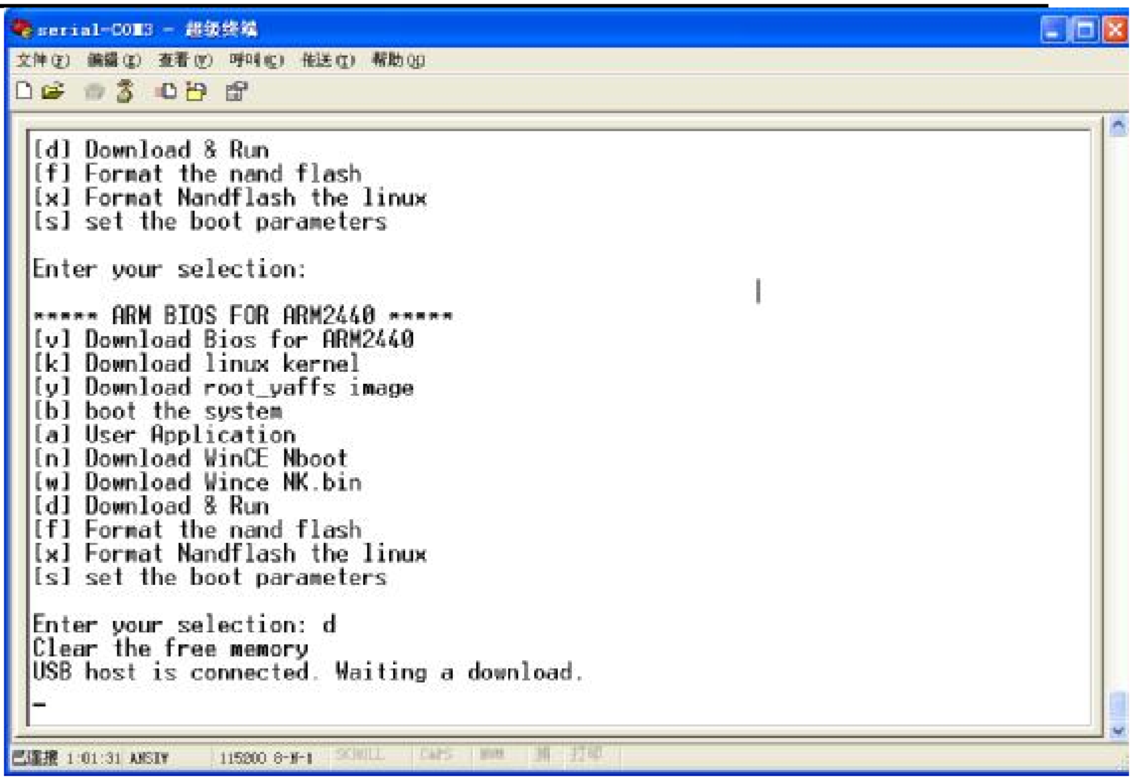
(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法),这时可以看到 DNW 的标题栏显示[USB: OK]，如果没有安装好驱动会显示[USB: x]，如图所示:



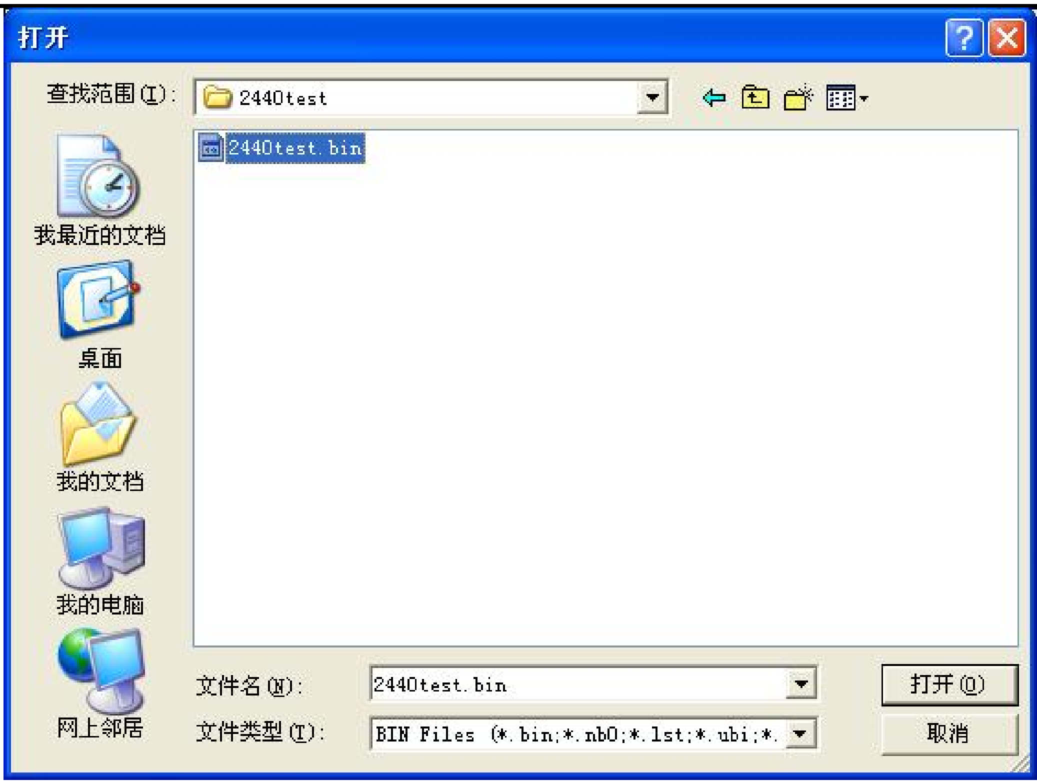
(3).点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000



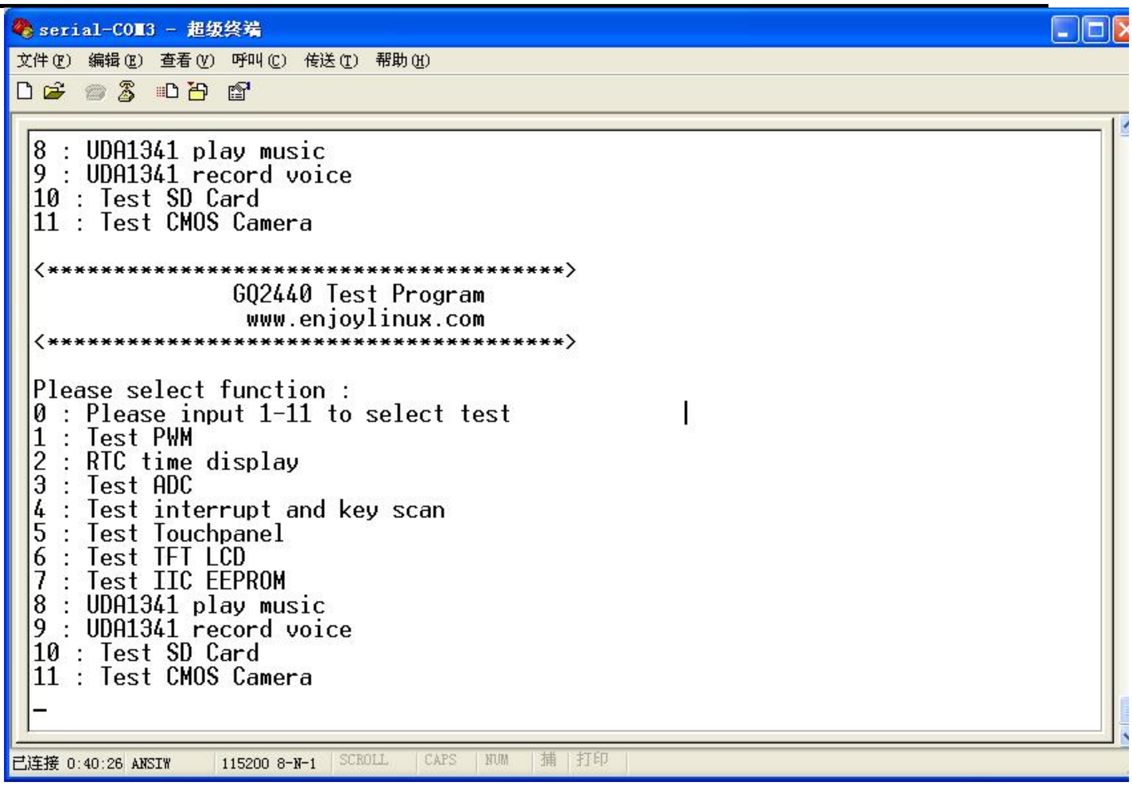
(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：



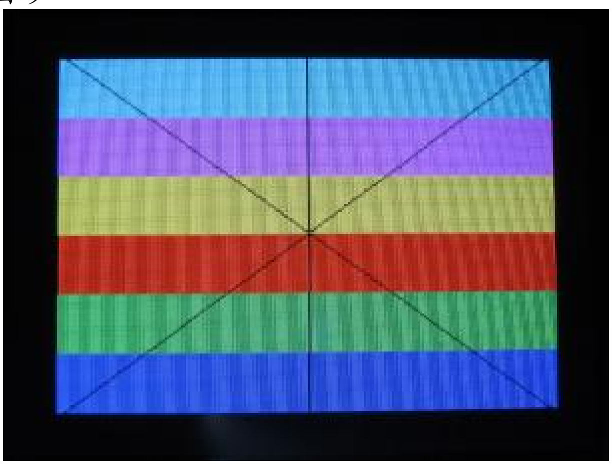
(5).点击 DNW 程序的“USB Port” “Transmit”，选择 2440test.bin 这个映象文件(在光盘的 images\2440test\目录下面)，接着点“打开”，这样就开始下载了。



(6)下载结束后，会自动运行，终端出现如下界面。


serial-COM3 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
8 : UDA1341 play music
9 : UDA1341 record voice
10 : Test SD Card
11 : Test CMOS Camera
<**>
GQ2440 Test Program
www.enjoylinux.com
<**>
Please select function :
0 : Please input 1-11 to select test
1 : Test PWM
2 : RTC time display
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test TFT LCD
7 : Test IIC EEPROM
8 : UDA1341 play music
9 : UDA1341 record voice
10 : Test SD Card
11 : Test CMOS Camera
已连接 0:40:26 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

LCD 出现彩带图形



当程序下载完成后，开发板会自动执行该测试程序。裸机测试程序也是一键式菜单。

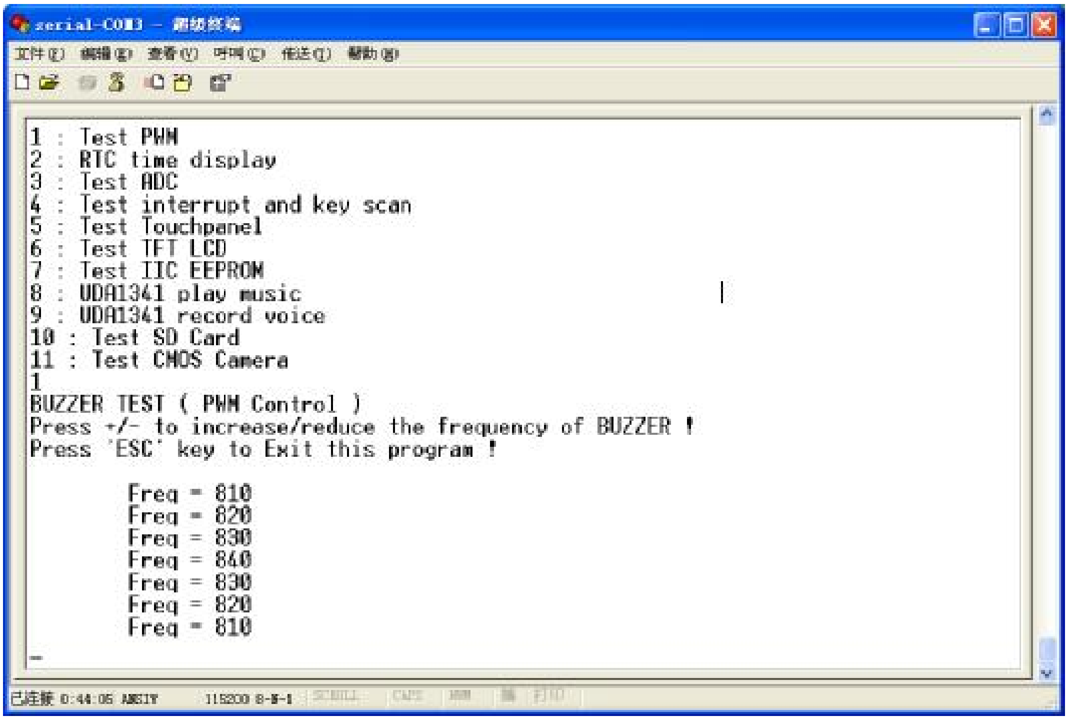
说明：2440test 运行起来之后，armvivi 就失去了对 CPU 的控制权，而 2440test 本身是没有 USB 部分支持的，因此此时会出现 USB 断开的情况。

2.3.2 外围资源测试

测试程序运行后，就可以进行相应的外围资源测试了，通过选择测试程序主菜单相应的选项，可以执行测试。

(1)蜂鸣器测试(Test PWM)

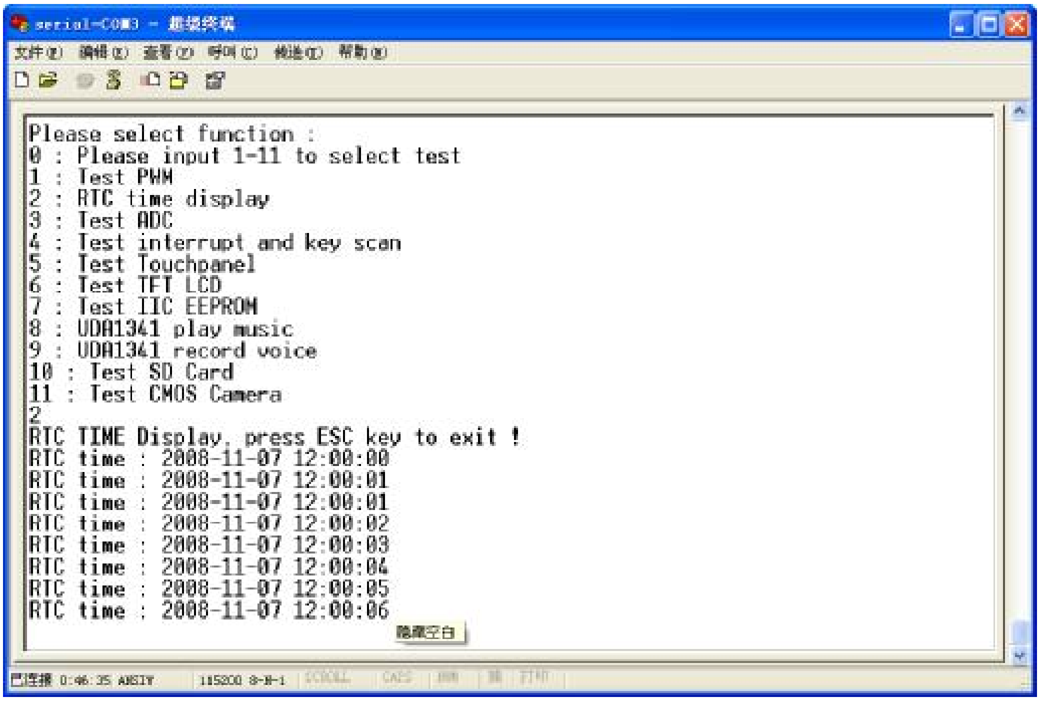
在主菜单中，输入“1”，再按“回车”键(即 Enter 键)，将开始进行蜂鸣器测试，蜂鸣器测试运行起来后，将会听到蜂鸣器发出声音。



按“-”键，蜂鸣器频率会降低，按“+”键频率升高，按“ESC”键可以推出该测试，并返回到主菜单中。

(2)实时时钟测试

在测试程序主菜单中，选择“2”，再按“回车”键，可以看到秒钟在不断的变化，这说明 CPU 的 RTC 在正常工作（注意：该时间并不是当前的时间，因为测试程序对其初始化并进行了赋值）

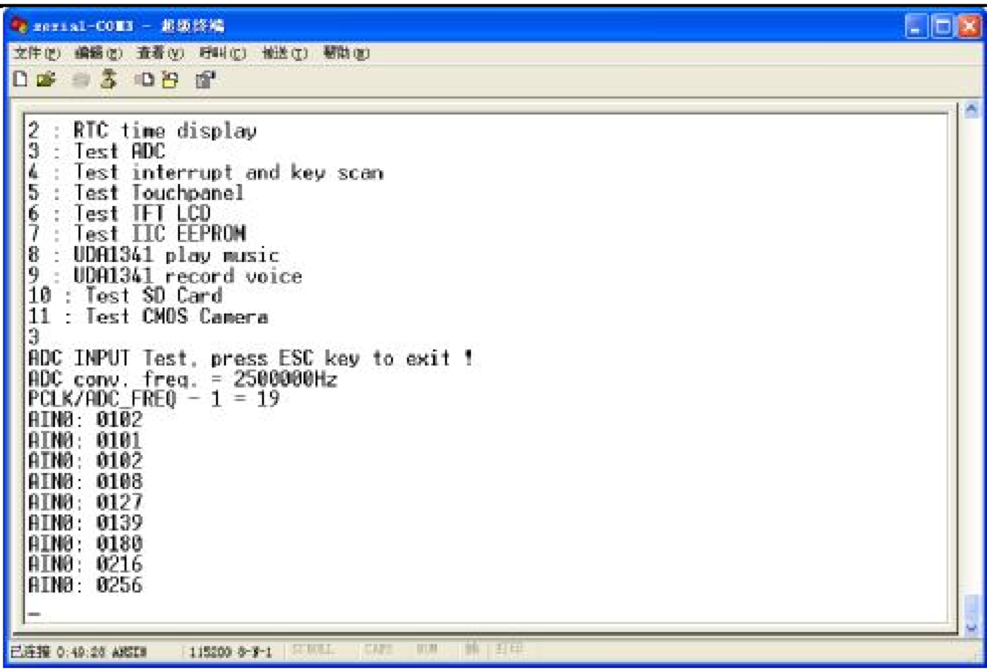


按“ESC”键退出该测试，并返回到主菜单。

(3)AD 测试

在主菜单中，输入“3”，再按“回车”键，开始执行 AD 测试。

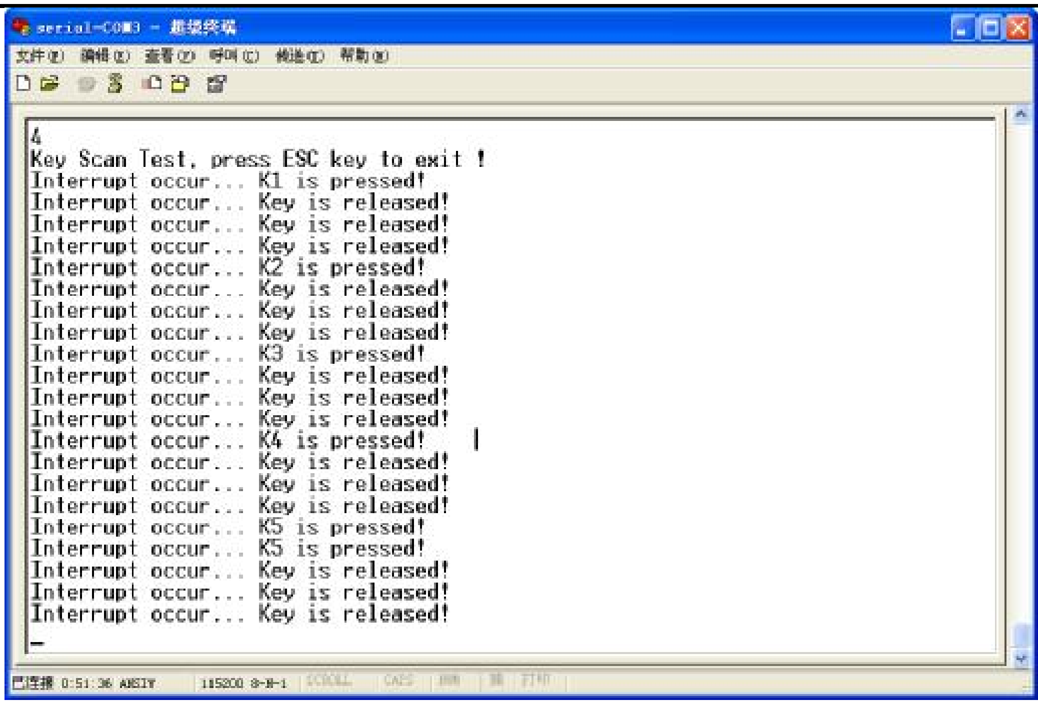
用户可以使用螺丝刀调节开发板上的 W1 或者 W2(这两个可调电阻接了 AIN0 和 AIN1)，可以看到 AD 的值在跟随调节电压在不断的变化。



按“ESC”键退出该测试，并返回到主菜单。

(4)按键测试

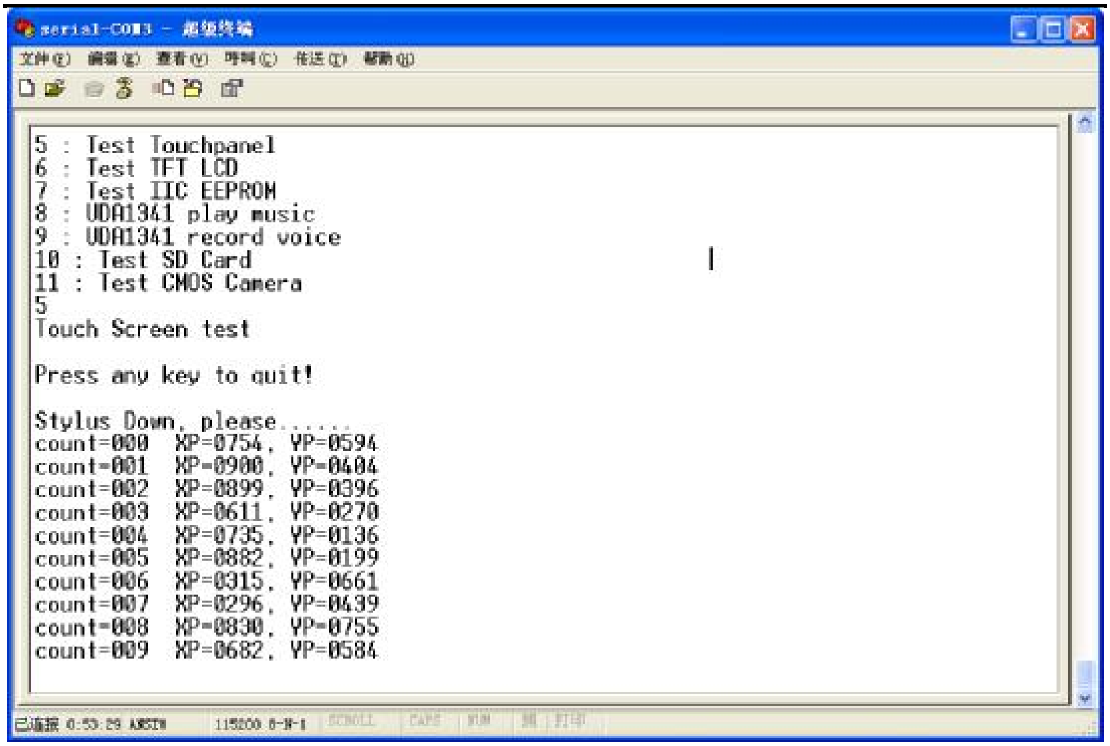
在主菜单中输入“4”，再按“回车”键，开始执行按键测试，此时按开发板上的 K1-K6 按键进行测试，可以看到串口终端打印相应的按键信息。



按“ESC”键退出该测试，并返回到主菜单。

(5)触摸屏测试

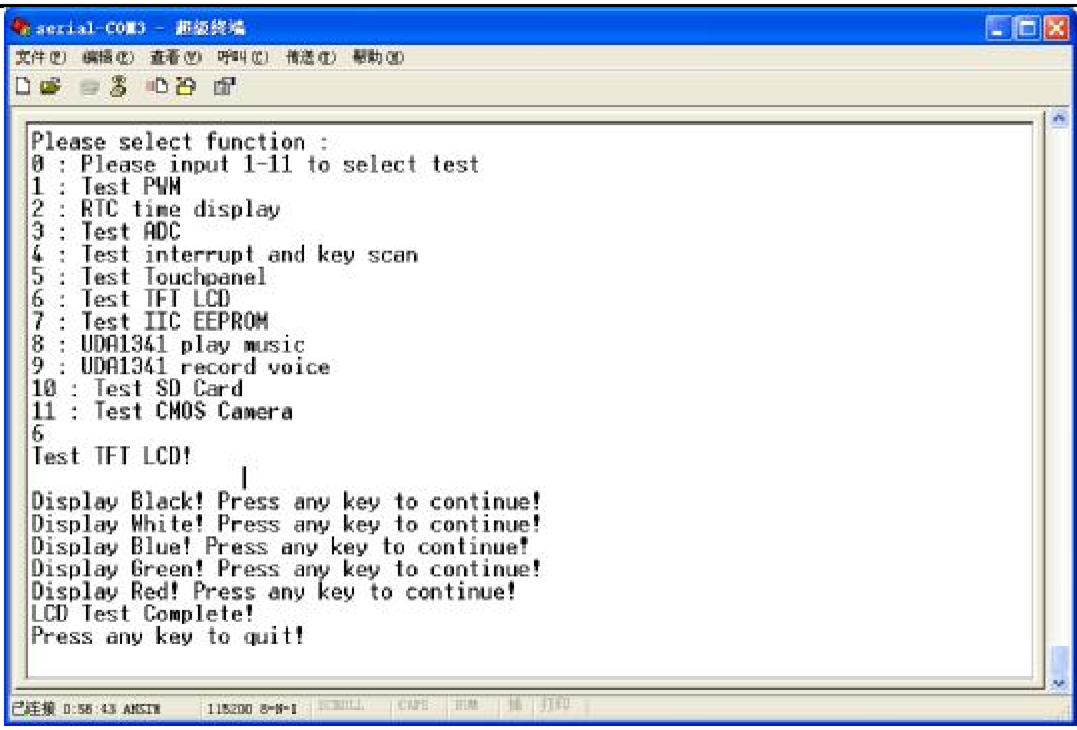
如果选购了 LCD 液晶屏，请使用附带的电缆连接开发板上的 LCD 接口。在主菜单中输入“5”，按“回车”开始进行触摸屏测试，这时用附带的触摸笔点击触摸屏，可以看到串口终端打印触摸点的坐标信息。



按“ESC”键退出该测试，并返回到主菜单。

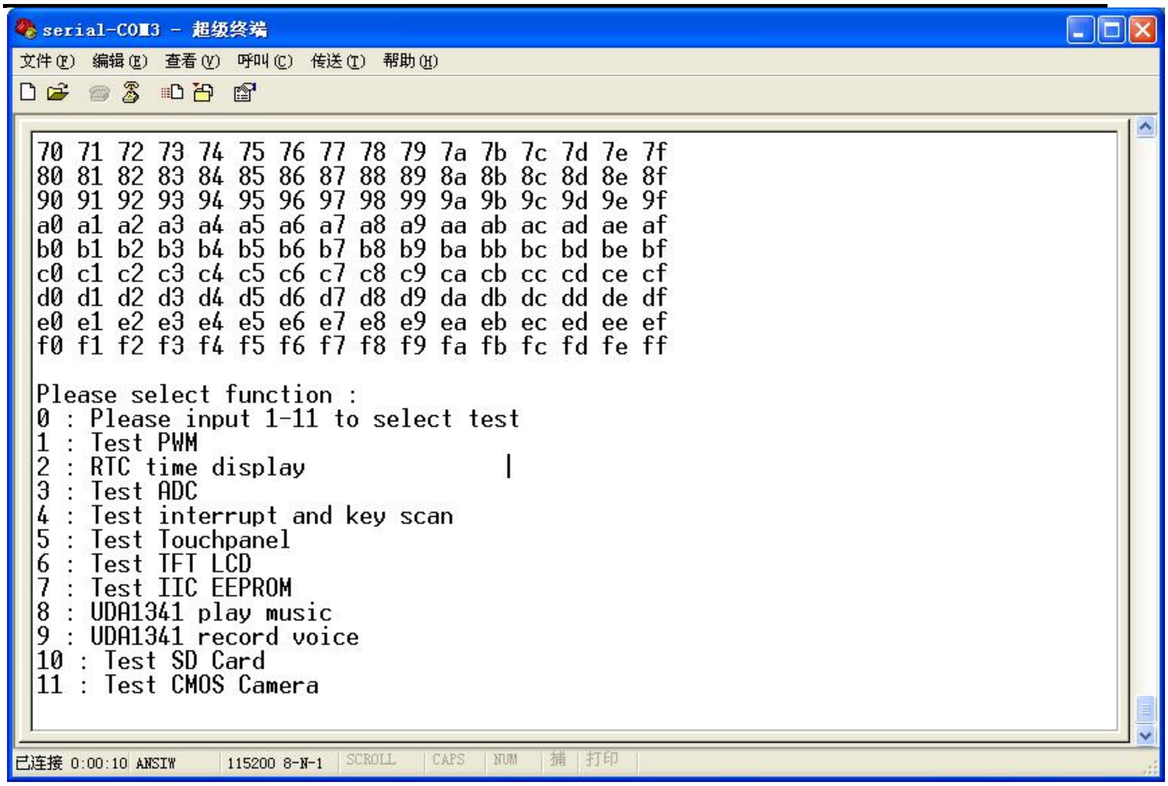
(6)LCD 或 VGA 模块输出测试

请烧写相应的 2440test 测试程序，主菜单中输入“6”按“回车”键开始执行测试，接着按照提示按任意键，LCD 将不断变化显示，直到最后显示一幅图片结束，并返回到主菜单。



(7)I2C 测试

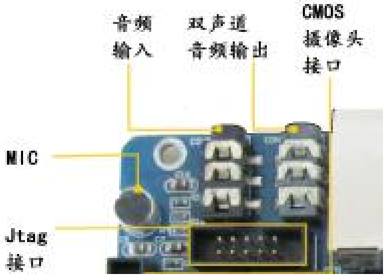
在主菜单中输入“7”，按“回车”键开始执行测试，程序将对 I2C 总线的芯片 AT24C08 进行读写，该测试主要是通过向 AT24C08 写入 0x-0xFF，然后读取出来。

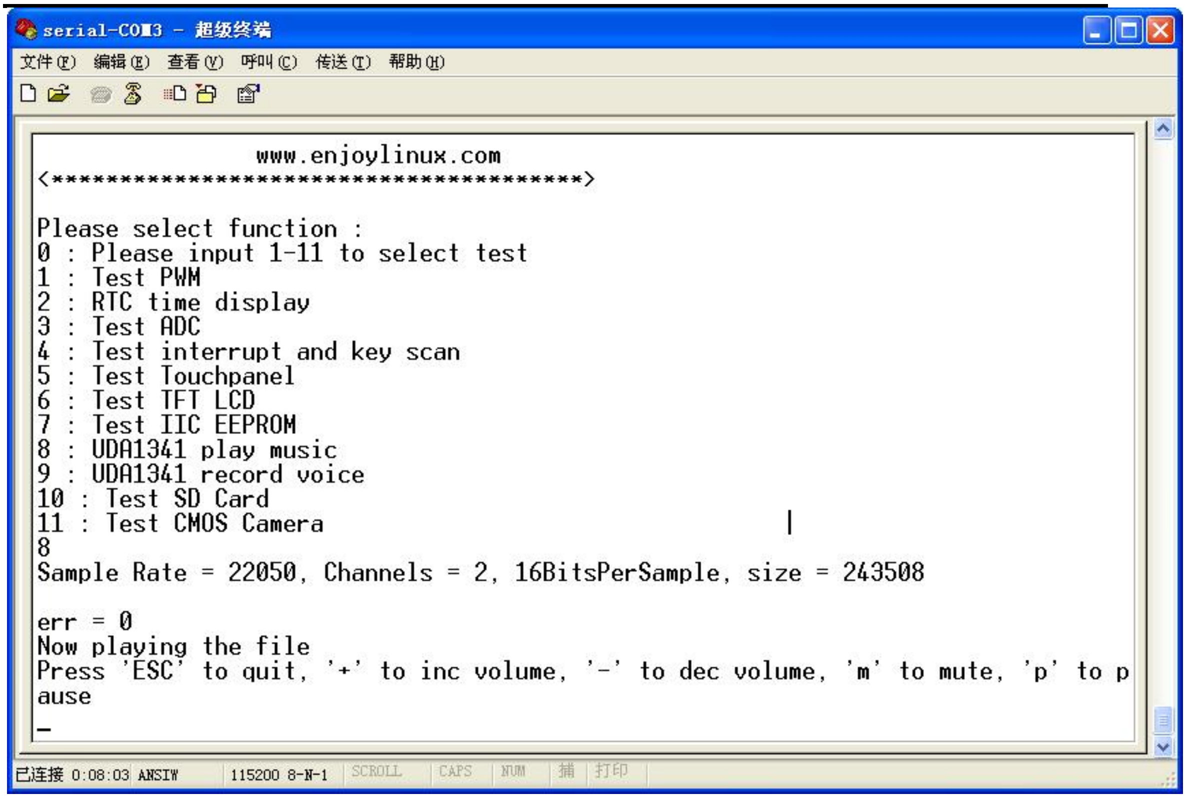


这个测试结束后，会自动回到主菜单。

(8)音频输出测试

先将音箱或耳机接到开发板的耳机孔座，如图中的【双声道音频输出】，在主菜单中输入“8”，按“回车”开始音频输出测试，这时可以从音箱听到 XP 的启动声音。

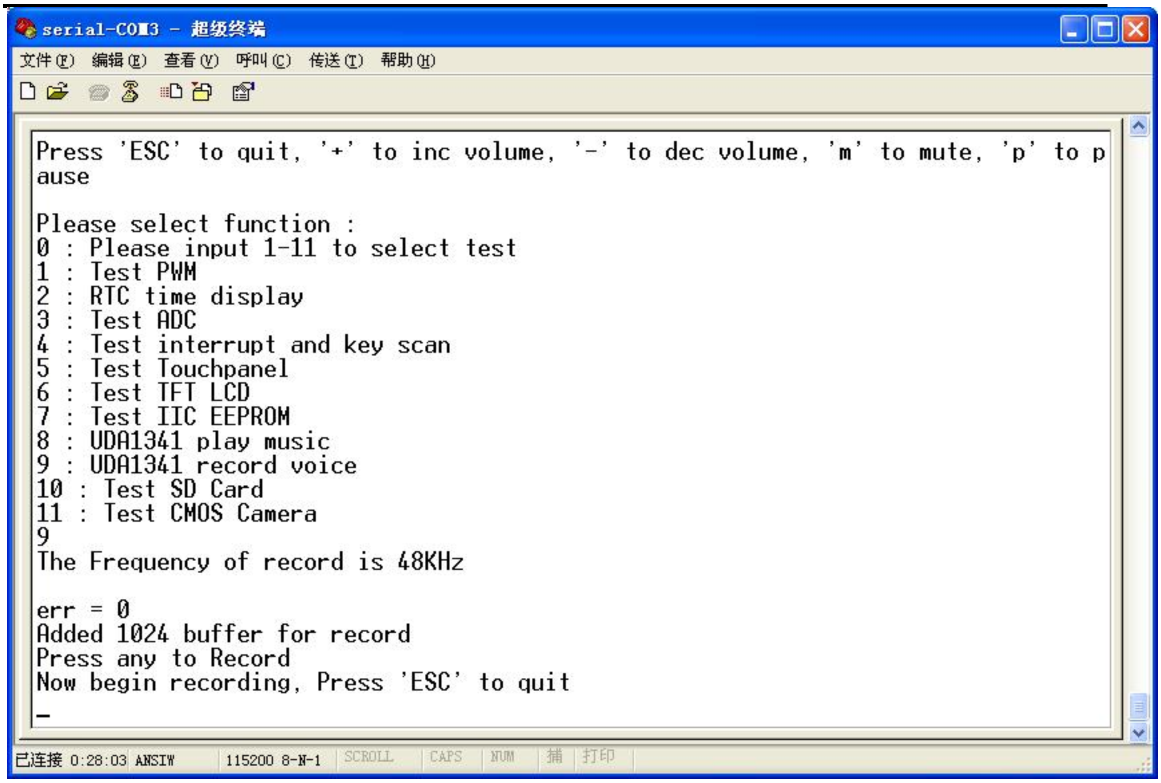




按"+"或者"-"可以增加或者减小音量，按"ESC"键退出测试，返回主菜单。

(9)音频输入测试

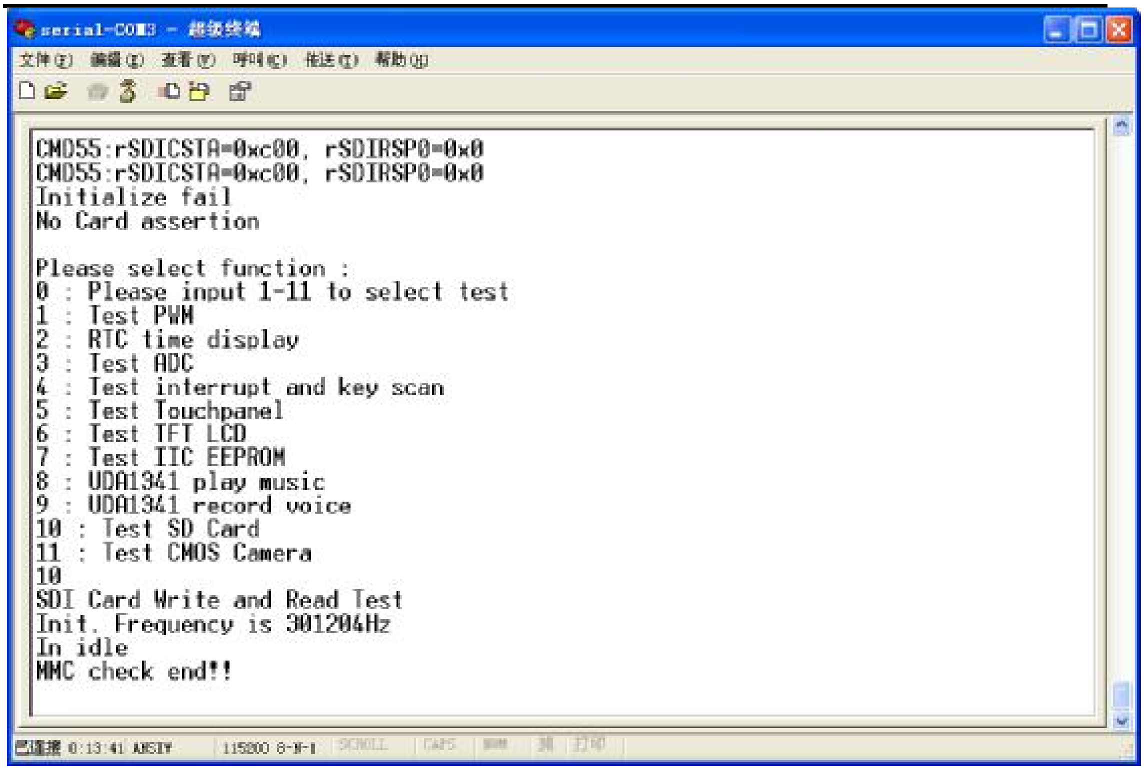
先将 MIC 接到开发板的音频输入座上，在主菜单中输入"9"，按"回车"可以开始录音测试。



(10)SD 卡测试

注意：**本测试会破坏 SD 卡中的数据，试用前请备份好 SD 卡中的数据。**

先将 SD 卡插入开发板的 SD 卡插座。在主菜单中输入“10”，按“回车”开始执行测试，程序将对 SD 卡进行读写，并出现如下界面：



待该界面显示 SD 卡读写成功，测试完毕，自动退回到主菜单。

2.4 初试 Linux 之图形界面 Qtopia 2.2.0 系统 (预装)

说明: Qtopia 2.2.0 是 Qt 公司基于 Qt/Embedded 2.3 库开发的 PDA 版(也是最终版)图形界面系统；自从 Qtopia 2.2.0 之后，Qt 公司就再也没有提供 PDA 版的图形系统了。最新版的 Qtopia 只有手机版本，而且 Qt 公司自从 2009.3 开始已经停止了所有 Qtopia PDA 版和手机版图形系统的授权，但依然继续开发 Qt/Embedded 库系统。

出厂之前，本开发板一般都预装了 Linux+Qtopia 2.2.0 图形界面，它包含了很多实用的小程序，拿到开发板后，你只要接上电源并开机就可以进行各项功能测试了，这不需要和电脑进行任何连接。

如果你使用的是 VGA 输出模块连接了显示器，还需要准备一个 USB 鼠标插到开发板的 USB Host 端口。

另外，本系统支持 USB 鼠标和触摸屏共存，并支持 USB 鼠标和键盘热插拔，你可以同时使用它们。

现在让我们先睹为快吧:

注意：**本小节内容的介绍均基于开发板 + 3.5” LCD 的屏幕截图。**

使用 3.5 寸平屏时，开机后会先后出现如下显示界面:



如果是重新安装会提示设置语言和时间信息:



设置时区：



设置时间：



Qtopia 桌面：

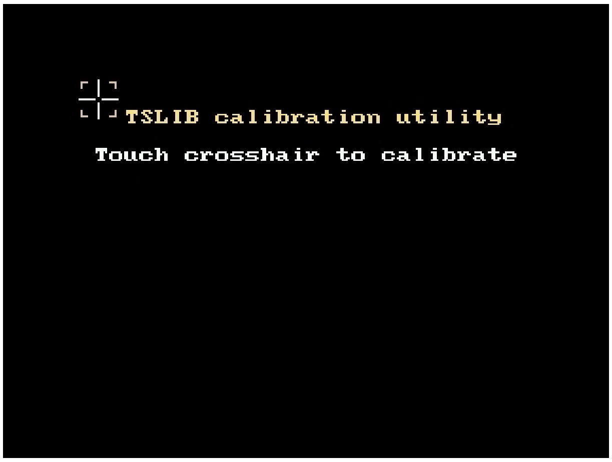


2.4.1 触摸屏校正

说明：如果按照下面的步骤没有校正准确，可以接上 USB 鼠标，在"设置"中打开"重校正"，再重新开始。

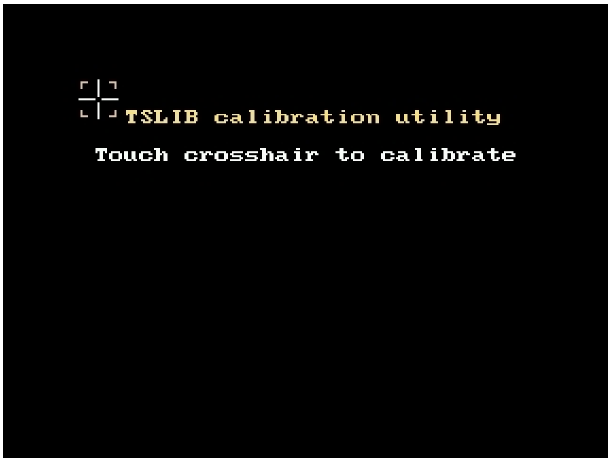
在两种情况下可以会出现触摸屏校正界面：

1. 如果你按照第三章的步骤重新安装了 Qtopia 系统，重启系统时首先出现如下触摸校正界面，依屏幕提示点击屏幕任何地方开始进行校正；然后依照屏幕提示，使用触摸屏逐步点击"十"型交叉点即可。



2. 进入系统后，点“开始->设置”切换到“设置”界面，再点“重校正”图标也会出现校正界面；然后依照屏幕提示，使用触摸屏逐步点击“十”型交叉点即可。





2.4.2 主要界面说明

进入 Qtopia 系统后主界面如下图所示:



可以看到 Qtopia 系统界面上方有四个图标，它们代表了四类程序/文档，单击任何一

个图标都可以进入相应的子类界面，它们都是类似的。另外点系统界面左下角的“开始”图标，也可以出现五个子类选择菜单，它们和系统界面上方的图标是对应的。

五个子类界面分别如下图所示，根据标题名称，其意自明。




应用程序
游戏
设置
Documents
十五子棋
打伞兵
打陨石
扫雷
纸牌
读心术
贪吃蛇
abc
13:23


应用程序
游戏
设置
Documents
浏览器
电源管理
网络
网络设置
语言
软件包
重校正
abc
13:23



2.4.3 播放 Mp3

在子类“应用程序”中单击“音乐”图标，出现播放器界面，在“Audio”列表中选择一首 mp3 歌曲文件，再点上方的“播放”按钮，开始播放 Mp3 文件。

说明：Audio 列表中的音频文件对应“Documents”子类中的所有有效音频文件。

提示：也可以在“Documents”中直接点击相应的文件名开始播放。







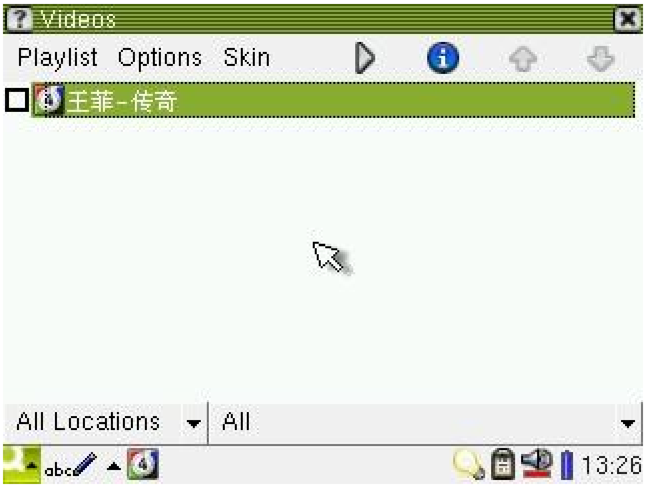
2.4.4 播放视频

在子类“应用程序”中单击“视频”图标，出现播放器界面，在“Video”列表中选择一个视频文件，再点上方的“播放”按钮，开始播放视频。

说明：Video 列表中的音频文件对应“Documents”子类中的所有有效视频文件。

提示：也可以在“Documents”中直接点击相应的文件名开始播放。


应用程序
游戏
设置
Documents
照相机
系统信息
终端
联系人
视频
计算器
语音便签
音乐
abc
13:25


Videos
Playlist Options Skin
王菲-传奇
All Locations
All
abc
13:26



2.4.5 图片浏览

在子类“应用程序”中单击“图片”图标打开图片浏览器，首先映入眼帘的是“文档”组中各个图片的缩略图，如果你插入了含有图片的 SD 卡或者优盘，其中的图片文件也会一并全部显示出来，下图是系统自带的 3 张图片，和插入含有其他图片的 SD 卡之后的截图。







Qtopia 2.2.0 系统的图片浏览器比以前的 Qtopia 1.7.0 有了很多改善，它可以对图片进行简单的编辑，而且使用起来也更方便，下面做一些简单的说明。

编辑图片

选择一张图片，单击打开，再点工具栏的笔形编辑按钮，进入编辑状态。

在编辑状态，点工具栏的彩色的圆形按钮，进行颜色调整，见下图。



旋转图片

在编辑状态，点工具栏的顺时针旋转按钮，可以依次以 90 度旋转图片。



剪切图片

在编辑状态，点工具栏的"剪刀"按钮，这时图片会被"蒙"上一层浅色的网，用触摸笔在这层网上任意区域画出一个"矩形"，抬起触笔，在所画的"矩形"上点击一下，这个区域就被单独选择出来了，这时你还可以对这个选区进行其他的调整。

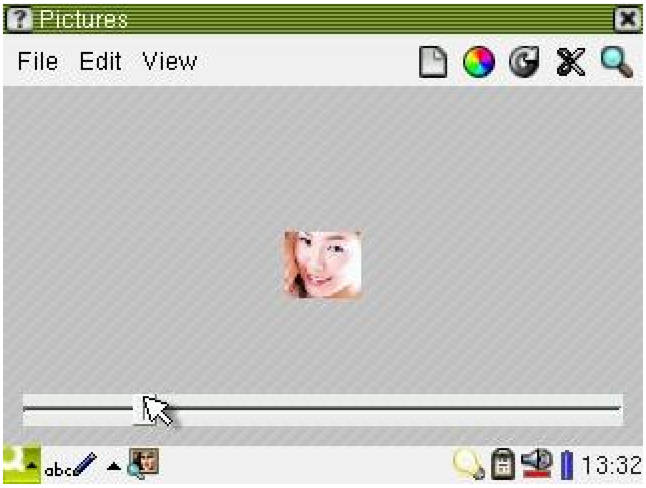




缩放图片

在编辑状态，点工具栏的"放大镜"按钮，可以实现对图片的缩放，如图。





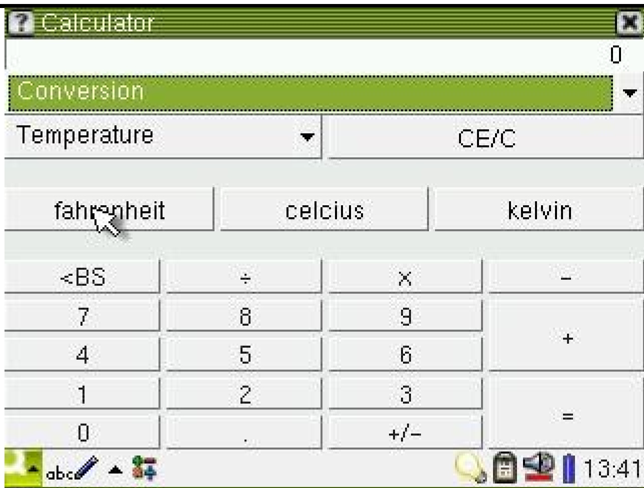
说明：“图片”浏览器列表中的图片文件对应“Documents”子类中的所有有图片文件。
提示：也可以在“Documents”中直接点击相应的文件名开始浏览。

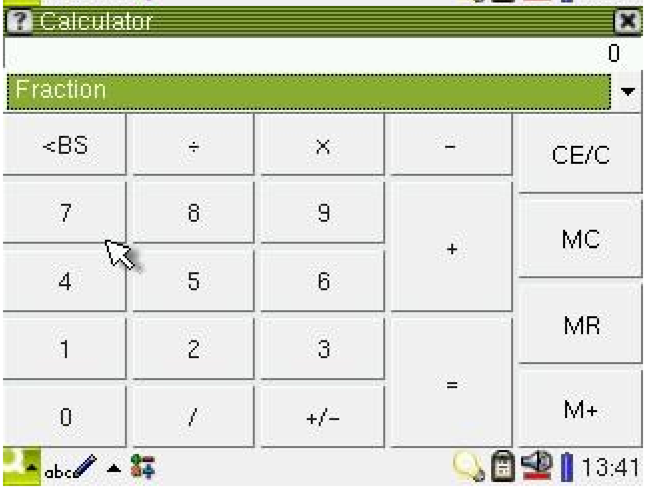
2.4.6 计算器

在子类“应用程序”中单击“计算器”图标，出现计算器界面，用户可以通过下拉列表选择 Simple, Fraction, Scientific, Conversion 等类型的计算器，如图：









2.4.7 命令终端

“终端”是 Linux 系统中通常用到的交互操作界面，通过“终端”可以运行很多 Linux 命令，查看系统信息等等。

有很多途径可以设置或打开一个“终端”：

在 Linux 系统启动的时候，可以把终端指向串口输出，这样就形成了串口终端，它的输入和输出都是通过串口进行的，无需图形界面，这是嵌入式 Linux 开发中最常用的方式。

在系统启动的时候，也可以把终端输出指向图形显示设备（如 LCD 或者 CRT 等），而把键盘设定为输入，这样就形成了一套独立的“输入输出系统”，它无需借助另外的 PC 即可操作。

当使用了图形显示设备，并且系统软件中增加了图形用户界面(GUI)时，就可以建立一个基于 GUI 系统的“命令终端窗口”，这时既可以通过标准的实体硬件键盘进行交互，也可以通过虚拟的“软键盘”进行交互，此处所讲的就是这种终端方式。

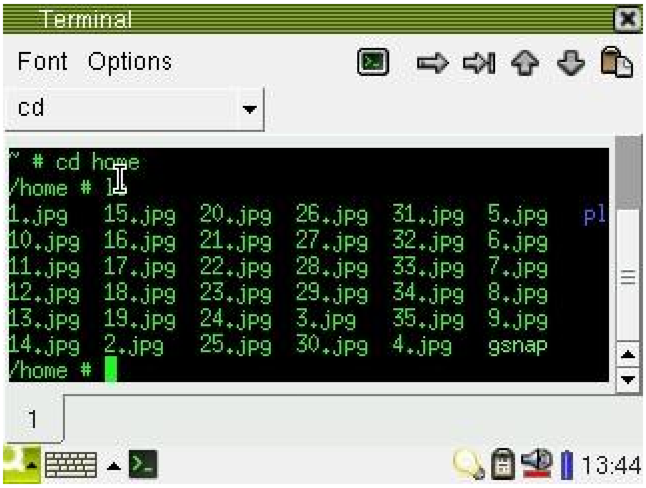
在子类“应用程序”中单击“终端”图标，出现命令终端窗口界面，此时可以接上 USB 键盘(不要在启动之前接 USB 键盘，否则不能使用)或者使用屏幕下方的软键盘输入 Linux 命令，你还可以点 Option 菜单中的某些选项进行设置，以改变显示的模式，如图。




Terminal
Font Options
ls
~ # ls
bin dev etc home
lib linuxrc lost+found mnt
opt proc root sbin
sys tmp usr var
www
~ #
13:43


Terminal
Font
Options
Hide Command List
Tabs on Top
Colors
Edit Command List
Green on Black
Black on White
White on Black
Default
Black on Red
Red on Black
Green on Yellow
Blue on Magenta
Magenta on Blue
Cyan on White
White on Cyan
Blue on Black
13:43





2.4.8 文件浏览器

在“设置”程序组中点“文件浏览器”图标打开它，如图：





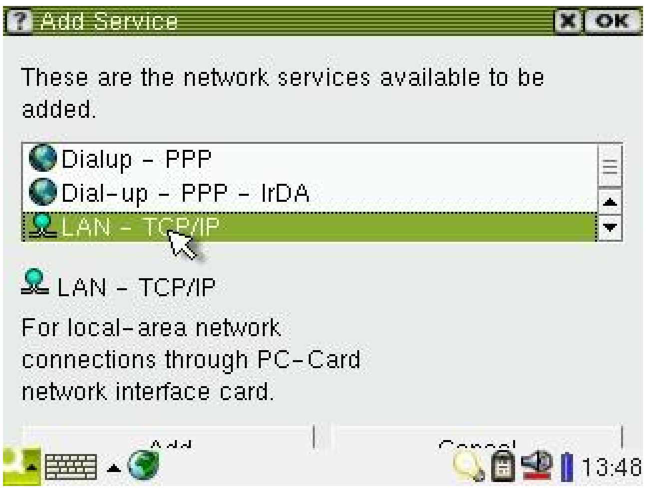
通过文件浏览器，你可以浏览管理开发板中的目录及文件。

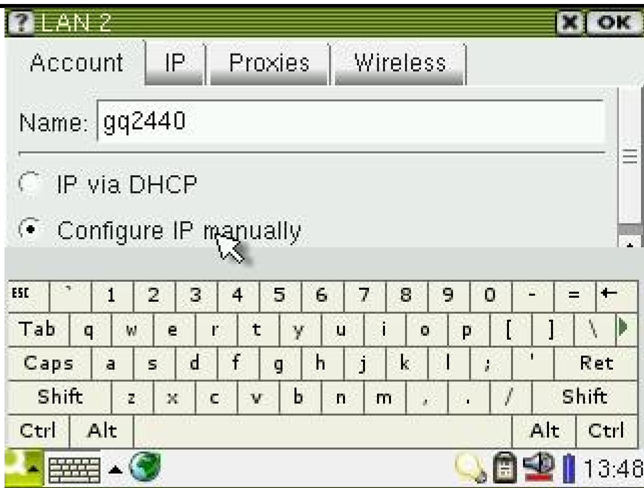
说明：Qtopia-2.2.0 原始版本是不带文件浏览器的，我们移植了 Qtopia-1.7.0 中自带的文件管理器，它们的功能和界面是完全相同的。

2.4.9 网络设置

在“设置”程序组中点“网络”图标打开它，设置方如图所示：









使用 qtopia 自带的这个网络设置有时不是很管用，这里还可以使用“设置”程序组中另一个“网络设置”程序。

在“设置”程序组中点“网络设置”图标打开它，设置方如图所示：



在这里，你可以进行常见的网络参数设置：

- 静态的 IP 地址 – 出厂缺省为 192.168.1.230
- 子网掩码 – 出厂缺省为 255.255.255.0
- 网关 – 出厂缺省为 192.168.1.1
- DNS 解析服务器 IP – 出厂缺省为 192.168.1.1，和网关地址相同
- 网卡的 MAC 地址 – 此地址由驱动程序通过软件设定，是可以修改的，本开发板出厂时所有 MAC 地址都是相同的，为 08:90:90:90:90:90

点"Save"按钮可以保存以上参数，并马上生效，重新启动开发板也可以保留此次的更改设定，与该设置程序相对应的参数文件为/etc/eth0-setting

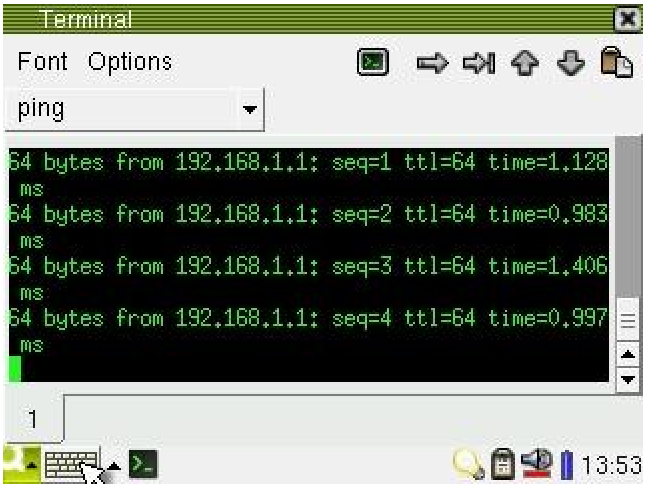
说明: /etc/eth0-setting 参数文件在系统重装后是不存在的，点“Save”按钮会自动生成；开发板出厂之前需经过测试，因此这个文件是存在的。另外，命令终端的 ifconfig 程序所作的 IP 地址更改对该配置文件没有影响。



2.4.10 ping 测试

请连接好开发板附带的网线，并设置好有效的网关，DNS 等参数，就可以通过 qtopia 的终端程序的 ping 命令来测试网络连通性了。在程序子类“应用程序”中点“终端”图标，打开相应界面，如下图，这里我们 ping 的是网关地址:





重要提示：**要 ping 互联网域名，必须要设置好正确有效的网关和 DNS，并且保证你的网络确实可以连通互联网。**

2.4.11 浏览器

在“设置”程序组中，点“浏览器”打开它，点开界面下方的软键盘，在界面上方的地址栏中输入一个网址，再点键盘上的“Ret”(回车)按钮，可以打开相应的网站。

说明：本开发板所用的网络浏览器为 Konqueror/Embedded，它是一个开发源代码的浏览器，具体的移植步骤见附录 1 中 Qtopia 的相关脚本(build-all 脚本中包含了移植该浏览器的所有步骤)。





2.4.12 串口助手

提示：**在使用该程序之前请连接好你需要测试的串口**

- 开发板上的CON1,23分别对应CPU 的UART0,1,2,其中UART0 已经转换为RS232电平，并通过DB9 插座COM0 输出，它在启动时已经被设置为console 终端，因此不能直接使用该程序测试。其他两个串口CON2,3 需要增加RS232 转换才可以和PC的串口连接使用。(本公司提供了型号为“OneCom”RS232 转换模块，如图)，连接PC时请注意你使用的串口线类型(交叉或直连)；不管你使用的是何种串口线，你也可以根据实际情况设置OneCom 模块上的COM.2 和COM.3 跳线。
- 本程序也支持市面上常见的USB转串口线，因为目前的笔记本大都没有串口，为了方便开发，我们的很多代理都提供了类似的转接线。把USB 转串口线接到开发板的USB HOST 端口上，你就可以扩充开发板的串口了。它对应的设备名一般为/dev/ttyUSB0,1,2, 3 等，这意味着你可以通过USB HUB扩展多个USB 转串口。

在“设置”程序组中点“串口助手”图标，打开相应的程序界面，如图：





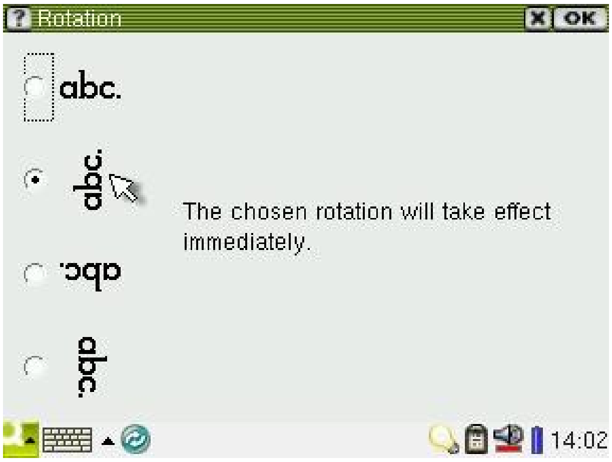
从该程序窗口的标题可以看到，默认设置为“ttySAC1 115200 8N1 [C]”，它表示默认端口的设置：

- 串口设备：/dev/ttySAC1，它对应 CPU 的第二个串口 UART1
- 波特率：115200
- 数据位：8
- 流控制：无
- 停止位：1
- [C]：表示字符模式，如果是[H]则表示 16 进制模式

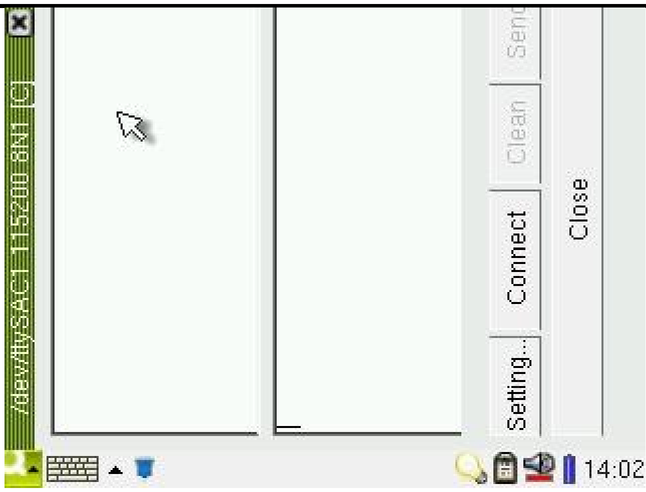
上图中有两个“编辑框”区域，上面的“编辑框”是用于显示接收到的数据，它实际上是不能编辑的；下面的“编辑框”可以通过 USB 键盘或者 Qtopia 的软键盘获取输入。

这里还有点小问题，程序的设置按键由于分辨率而超出屏幕显示范围之外，这是可以在“设置”程序组中点“旋转”图标，设置屏幕显示方式，方法如下：





这时再打开源程序便可看到整个程序界面:



先点击【Setting】，进入设置界面，按照如图方式设置，这时串口 0 成为了通讯端口，点击【OK】保存设置退出。



设置参数说明:

Comm Port: 可以选择 CPU 自带的/dev/ttySAC0,1,2 三个串口;也可以选择 USB 转串口对应的/dev/ttyUSB0,1,2,3

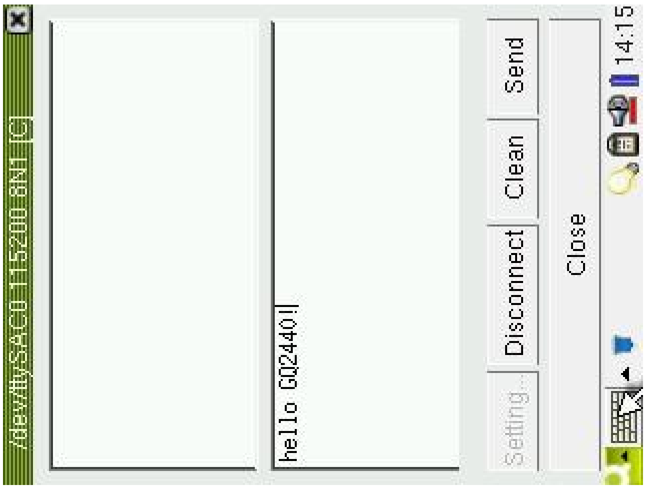
Speed: 可以选择各种常见的波特率

Data: 可以选择数据位是 8 位/7 位，常见的是 8 位

Hex: 此项表示采用 16 进制输入或者显示数据。

点 Connect 按钮，以打开开发板串口/dev/ttySAC0，在窗口下面的编辑框输入一些字符，点 Send 按钮，就可以向与它相连的串口设备发送数据了，下图显示的是通过 Windows 超级终端接收的数据截图(注意：与此终端对应的串口也应该设置为 115200 8N1)。





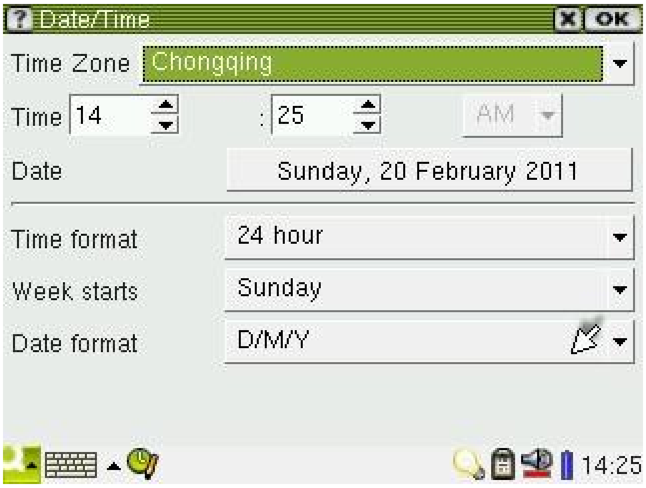


2.4.13 设置时区-日期-时间-闹钟

本开发板在出厂的时候日期是不准确的,不过你可以自己调整一下,因为 CPU 内置了实时时钟(RTC),开发板上有备份电池,所以调整后的时间在关机后是可以"保存"的,当然这需
要相应的驱动支持了，调整时间的步骤如下:

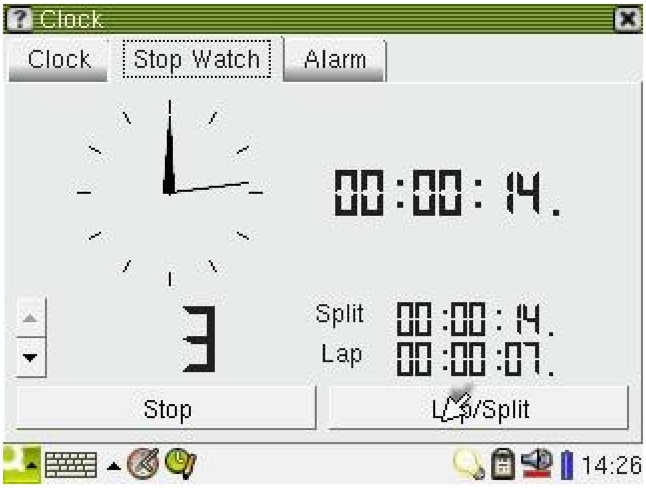
点击任务栏右下角时间显示区域，会出现一个菜单，在其中选择"Set time…"，打开时间设置界面，如图，在此界面你可以设置时区，调整日期和时间等。





在菜单中选择“Clock...”，出现钟表界面，点其中的“Stop Watch”打开“秒表”程序。


Clock
Clock
Stop Watch
Alarm
14:26
Sunday, 20 February 2011
14:26


Clock
Clock
Stop Watch
Alarm
00:00:14.
3
Split
00:00:14.
Lap
00:00:07.
Stop
Lap/Split
14:26

另外你还可以设置闹钟，如图，闹钟时间到，从音频输出可以听过“嘀嘀”声，它会持续一分钟，并出现如下界面，点“OK”可以结束闹钟声。

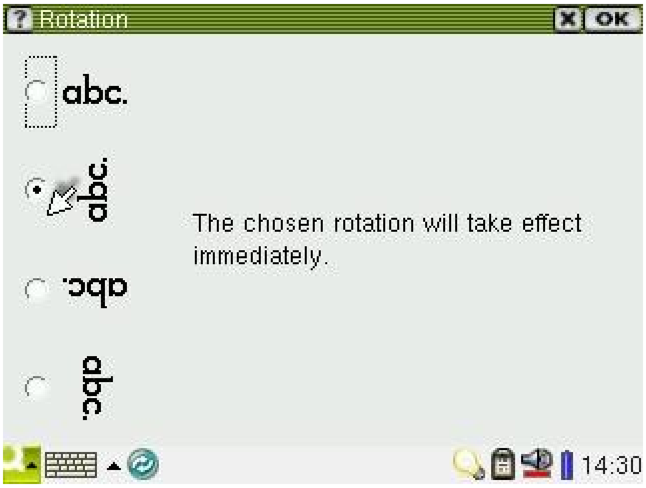




2.4.14 屏幕旋转

在“设置”程序组点“旋转”图标，进入相应的界面，如图，你可以选择 4 个界面旋转方向，如图。





选择好你想要的方向，点“OK”返回，即可看到结果，如图。

说明：有时需要重新启动 Qtopia 系统才可以看到旋转的效果，因为这个程序是 Qtopia

自带的，为保持其代码原始性，我们没有去修改它。另外，旋转的效果完全是 Qtopia 软件实现的，与 LCD 的底层驱动无关。



2.4.15 关于关机

在"设置"程序组中，有个"关机"图标，打开它，如图，其中有四个关机选项：

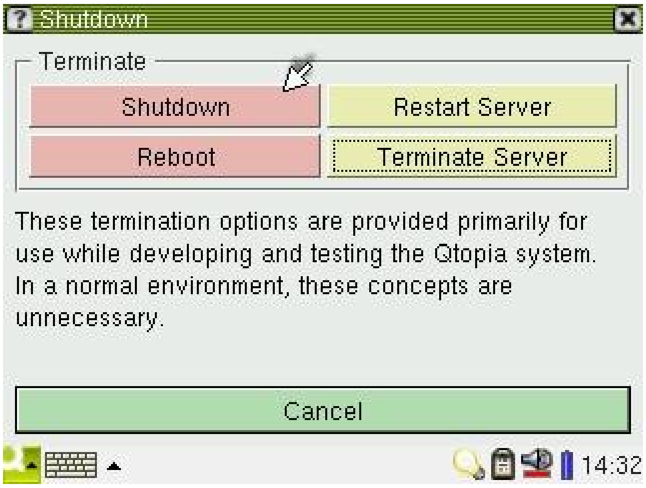
Shutdown：按下此按钮，Linux 系统就会逐项关闭各个程序和服务，直到整个系统关闭，这时 CPU 已经完全不工作了，此时整个系统的功耗是最低的。因为本开发板并无相应的硬件关机电路，因此你仍然可以看到板上的电源灯在亮。

Reboot：这是"热"重启按钮，如果你使用的是 NOR FLASH 模式，系统会逐项关闭各个应用和服务，重启后会停留在 Supervivi 的菜单模式；如果你使用的是 NAND FLASH 模式，系统会逐项关闭各个应用和服务，然后自动重启重新进入 Qtopia 系统。

Restart Server: 是指重新启动 Qtopia 图形系统，此时并不会影响到基本的 Linux 系统；

Terminates Server: 是指关闭 Qtopia 图形系统，点击它之后图形界面就完全失效了，此时屏幕上的显示是遗留在内存中的数据，并不是有效的图形系统界面。





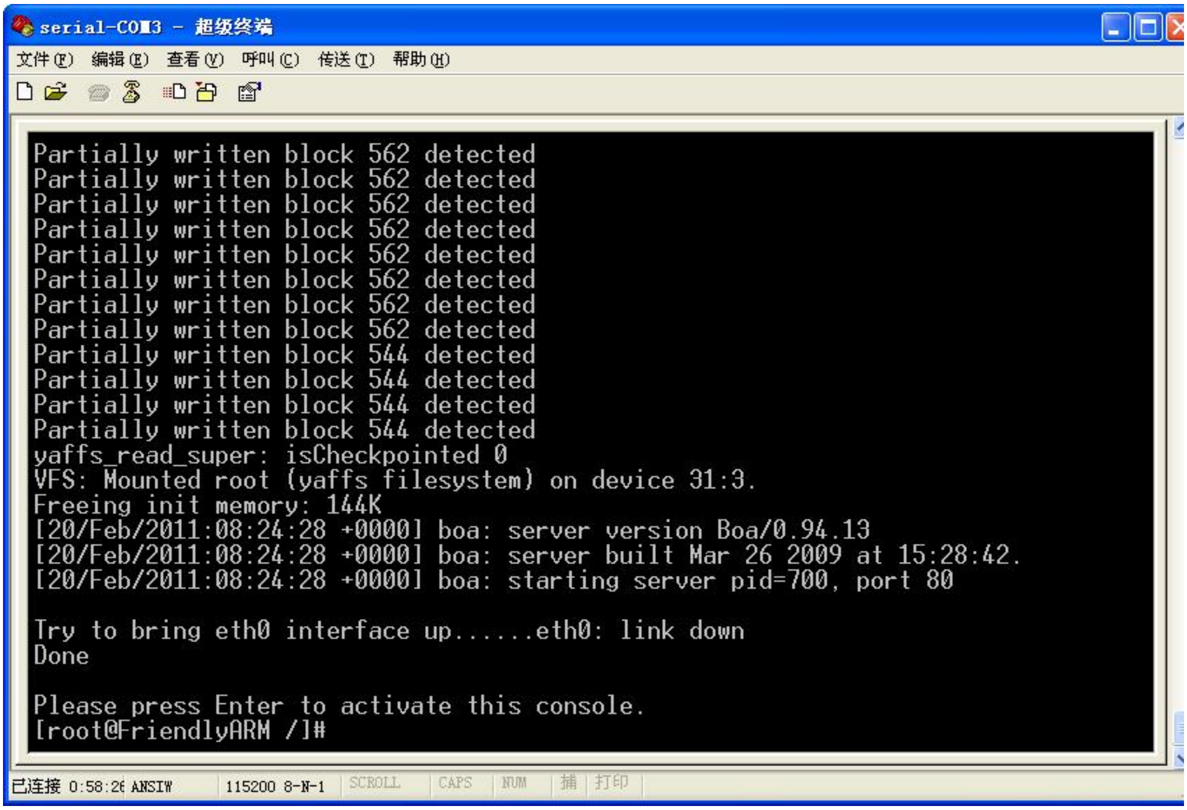
2.5 通过串口终端操作开发板

说明：每个期望学习 Linux 的嵌入式爱好者都应该学会熟练使用终端控制台的操作，所有平台的 Linux 指令都是相似的，超过 99%的命令是相同的；当你能够在命令行运指

如飞的时候，你就已经深深爱上了 Linux。

进行本小节的操作之前，请先按照 2.1.3 章节正确设置好超级终端。

下图是通过串口终端显示的 Linux 登录界面，实际可能与此不完全相同，但基本都是类似的，根据提示，按下回车，就可以开始 Linux 控制台之旅了。



2.5.1 播放 mp3

测试程序名称：madplay

源代码包的名称：Linux 源代码和工具\madplay-0.15.2b.tar.gz

交叉编译器：arm-linux-gcc-4.3.2 with EABI

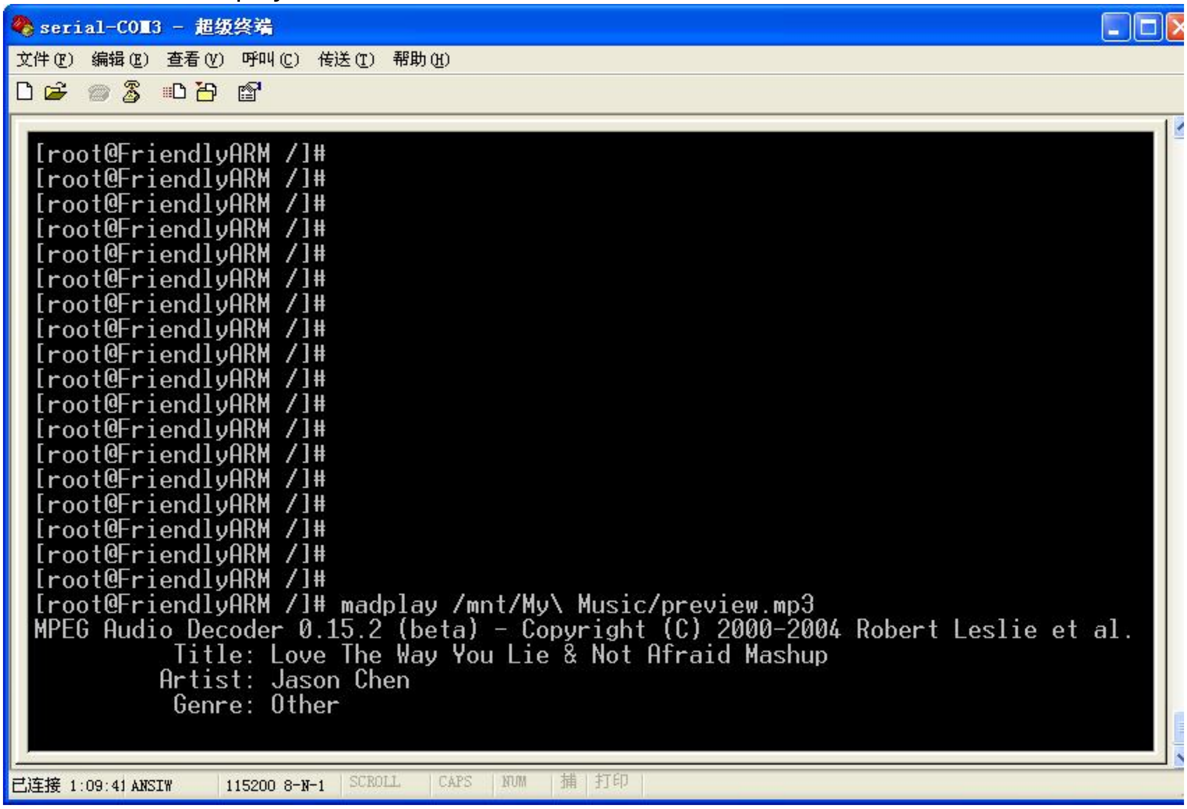
madplay 是我们移植的一个基于控制台下的 mp3 播放器。它有多种播放控制模式，最简单的使用方法是:

#madplay your.mp3

该命令将以缺省模式播放 your.mp3 文件(开发板中并无 your.mp3 文件，这里只是举例说明)。

注意：系统启动后将会自动播放当前目录下的 shanghaitan.mp3 文件。

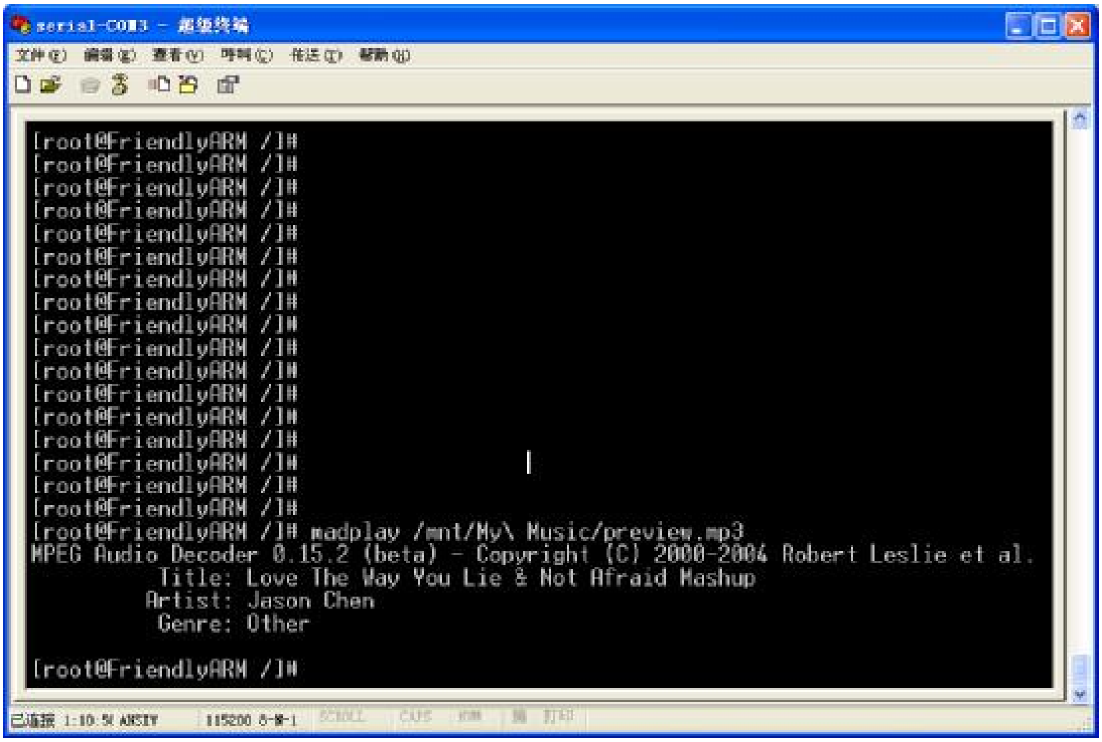
可以运行“madplay -h”查看其使用帮助。



2.5.2 如何中止程序的运行

要中止程序的运行，可以在终端控制台下同时按下**Ctrl+c**，注意：先按**Ctrl**，不要放开，再按下**c** 键即可。

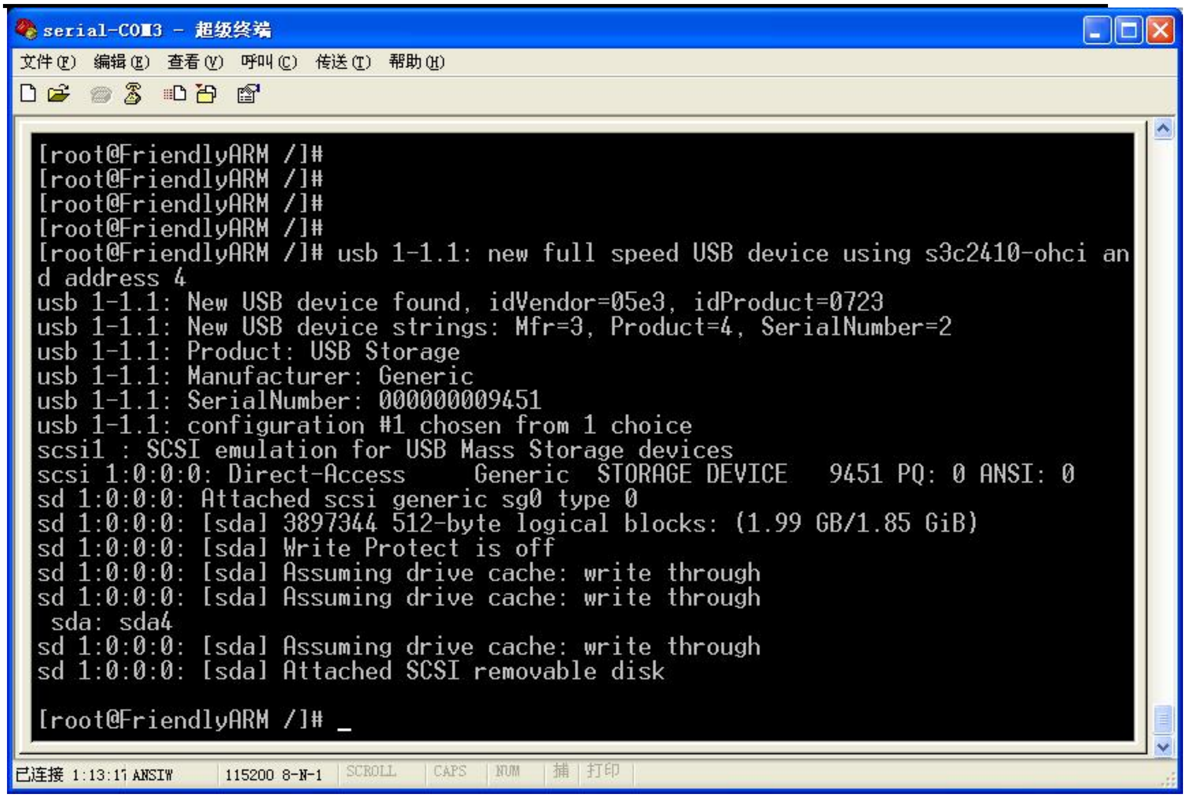
例如：我们刚刚使用madplay 命令播放了mp3，如果要中止这个程序的运行，可以按下**Ctrl+c** 键。



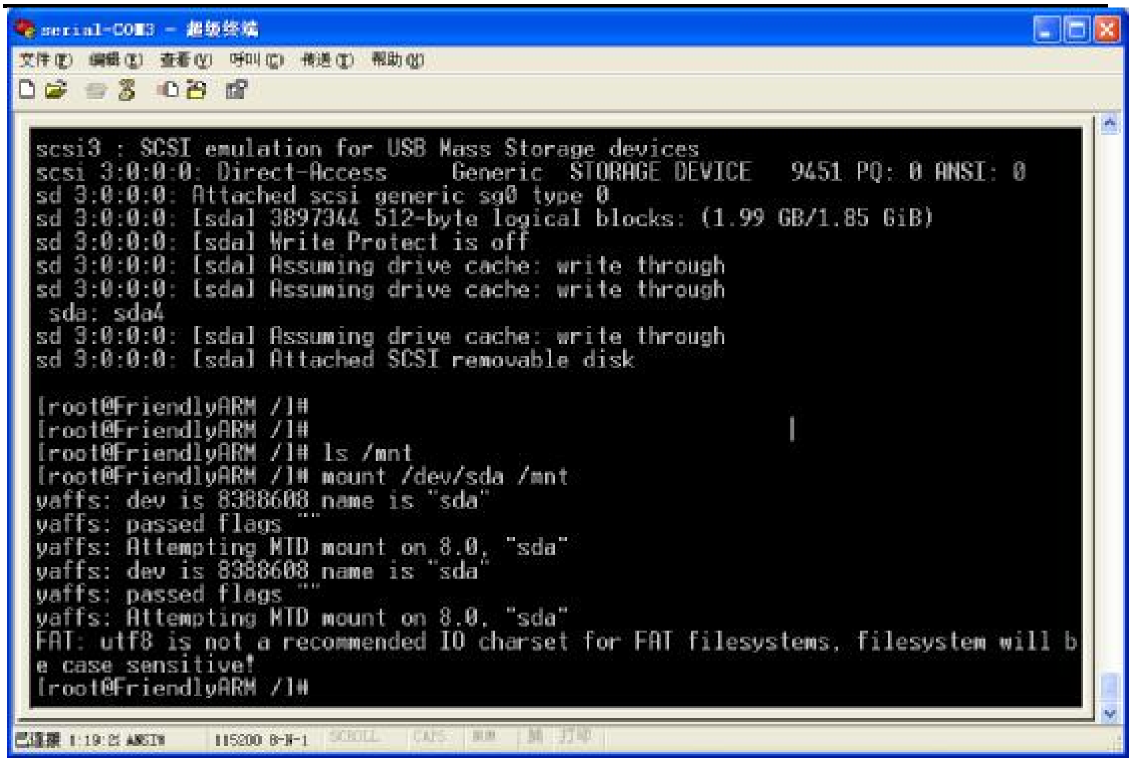
另外，如果程序是在后台运行，可以使用 **kill** 命令杀掉该进程

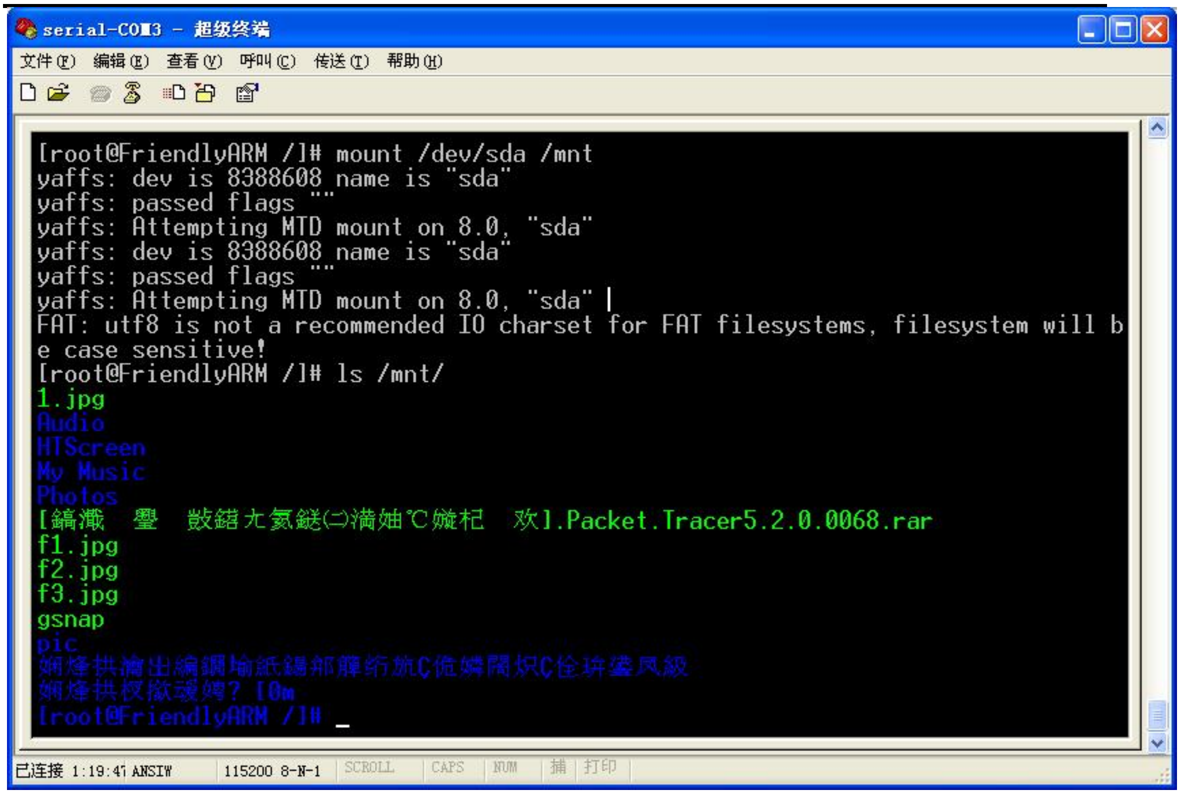
2.5.3 使用优盘/移动硬盘

插入优盘之后，在串口会出现类似如下信息：



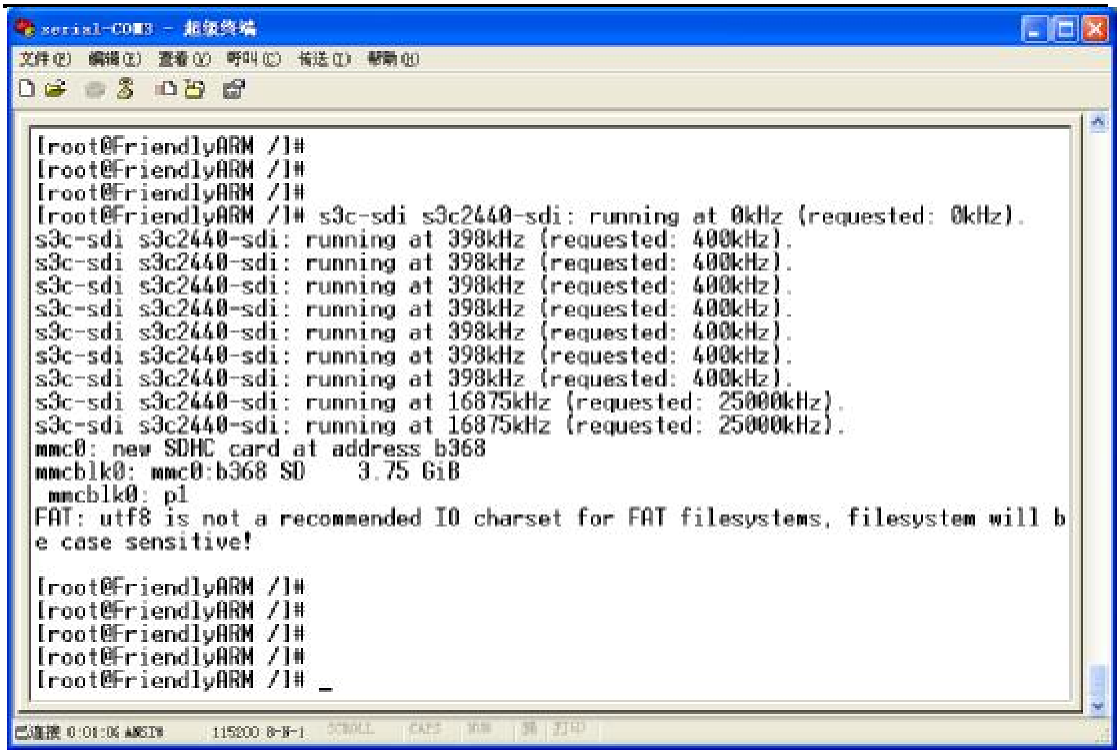
从打印信息中可见，这个 U 盘设备名称是 sda，可以通过命令：mount /dev/sda /mnt 将此设备挂在到/mnt 目录访问：


serial-COM3 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
scsi3 : SCSI emulation for USB Mass Storage devices
scsi 3:0:0:0: Direct-Access   Generic STORAGE DEVICE   9451 PQ: 0 ANSI: 0
sd 3:0:0:0: Attached scsi generic sg0 type 0
sd 3:0:0:0: [sda] 3897344 512-byte logical blocks: (1.99 GB/1.85 GiB)
sd 3:0:0:0: [sda] Write Protect is off
sd 3:0:0:0: [sda] Assuming drive cache: write through
sd 3:0:0:0: [sda] Assuming drive cache: write through
sda: sda4
sd 3:0:0:0: [sda] Assuming drive cache: write through
sd 3:0:0:0: [sda] Attached SCSI removable disk
[root@FriendlyARM /]#
[root@FriendlyARM /]#
[root@FriendlyARM /]# ls /mnt
[root@FriendlyARM /]# mount /dev/sda /mnt
yaffs: dev is 8388608 name is "sda"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 8.0, "sda"
yaffs: dev is 8388608 name is "sda"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 8.0, "sda"
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!
[root@FriendlyARM /]#
已连接 1:19:21 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

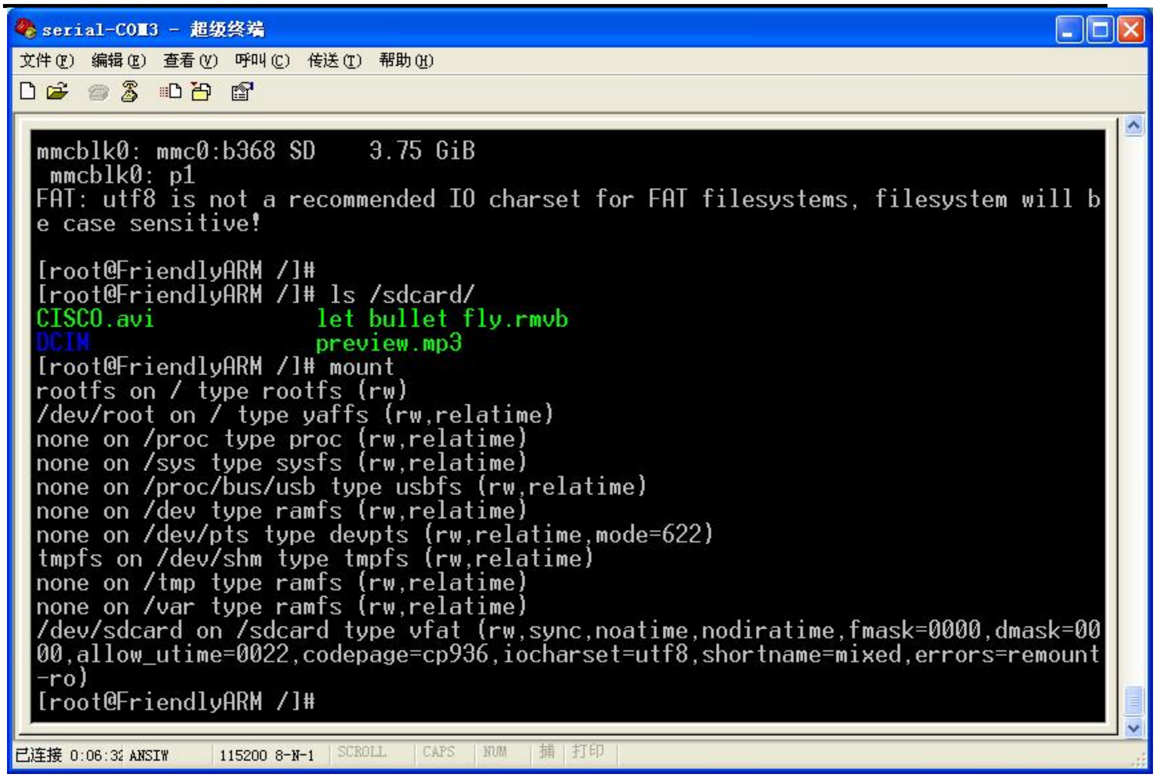


2.5.4 使用 SD 卡

SD 卡可以自动识别并挂载，插入 SD 卡之后可以在串口看到如下信息:



系统会会自动创建/sdcard 目录，并把 SD 设备挂载到上面，如图：



2.5.5 如何通过串口与 PC 互相传送文件

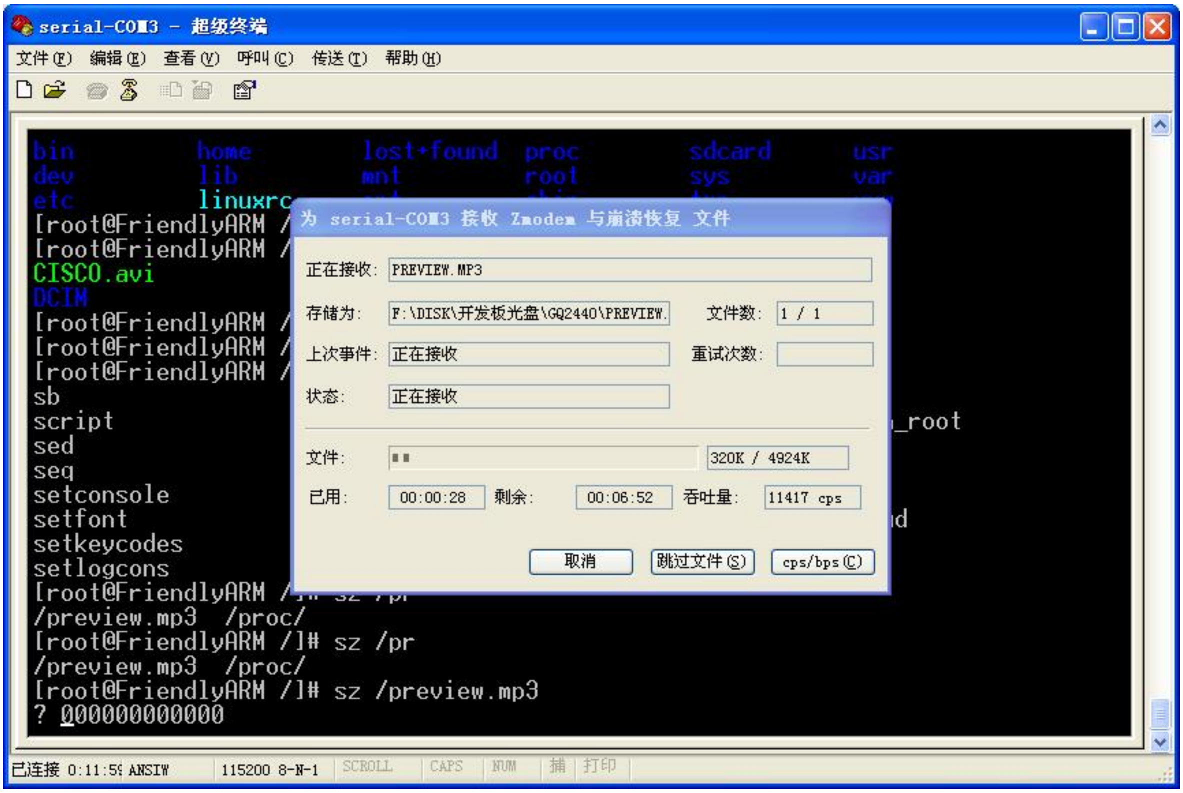
当通过串口终端登录系统之后，可以使用 rz 或者 sz 命令通过串口与 PC 互相传送文件，具体操作如下。

(1)使用 sz 向 PC 发送文件

在超级终端窗口中，点鼠标右键，在弹出的菜单中选择“接收文件”开始设置接收文件目录和协议，如图所示。



然后在终端的命令行输入"sz /shanghaitan.mp3"命令，开始向 PC 传送位于"/"目录的 shanghaitan.mp3 文件(或者其他文件，改一下路径和文件名就可以了)，因为该文件比较大，所以需要多等几分钟，发送完毕，系统会自动保存文件到您设置的目录里面，如图。



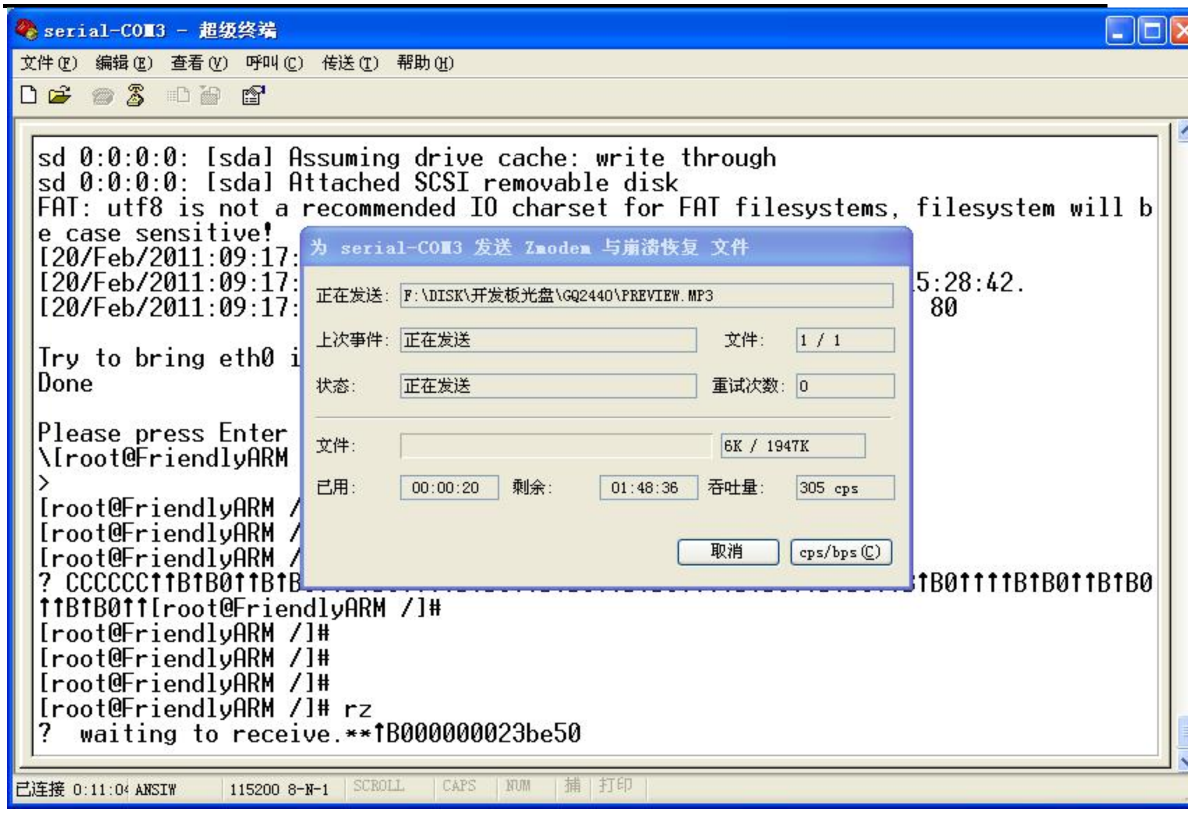
(2)使用 rz 命令下载文件到开发板

在串口终端输入"rz"命令，开始接收从 PC 传过来的文件。

然后在超级终端窗口中，点鼠标右键，在弹出的菜单中选择"发送文件"，设置好要发送的文件和使用的协议，如图所示，开始向开发板发送文件。



点"发送"，开发板开始接收文件，如图所示。



2.5.6 控制板上的 LED

测试程序名称：led-player led

源代码包的名称：led-player.c led.c

源代码在光盘中的位置：Linux 源代码和工具\examples-20100108.tar.gz

交叉编译器：arm-linux-gcc-4.3.2 with EABI

开发板上对应的设备名：/dev/leds

对应的内核驱动源代码：Linux-2.6.32.2/drivers/char/GQ2440_leds.c

(1) LED 服务器

开机进入系统后，将会自动运行运行一个 LED 服务程序(/etc/rc.d/init.d/leds)，它其实是调用了 led-player 的一个脚本，led-player 开始运行后，将会在/tmp 目录下创建一个 led-control 管道文件，向该管道发送不同的参数可以改变 led 的闪烁模式:

#echo 0 0.2 > /tmp/led-control

运行该命令后，4 个用户 led 将会以每个间隔 0.2 秒的时间运行跑马灯。

#echo 1 0.2 >/tmp/led-control

运行该命令后，4 个用户 led 将会以间隔 0.2 秒的时间运行累加器。

#/etc/rc.d/init.d/leds stop

运行该命令后，4 个用户 led 将会停止闪动。

#/etc/rc.d/init.d/leds start

运行该命令后，4 个用户 led 将会重新开始闪动。

(2) 单独控制LED

/bin/leds 是一个可以控制单个 led 的实用程序，要使用 leds 必须先停止 led-player，如下命令:

#/etc/rc.d/init.d/leds stop

该命令将停止 led-player 对 led 的操纵。led 的使用方法如下:

[root@fa /]# led

Usage: leds led_no 0|1

led_no 是要操作的 led(可为 0，1，2，3)，0 和 1 分别代表关闭和点亮。

#led 2 1

将点亮 LED3

2.5.7 测试板上的按键

测试程序名称：buttons

源代码包的名称：Buttons_test.c

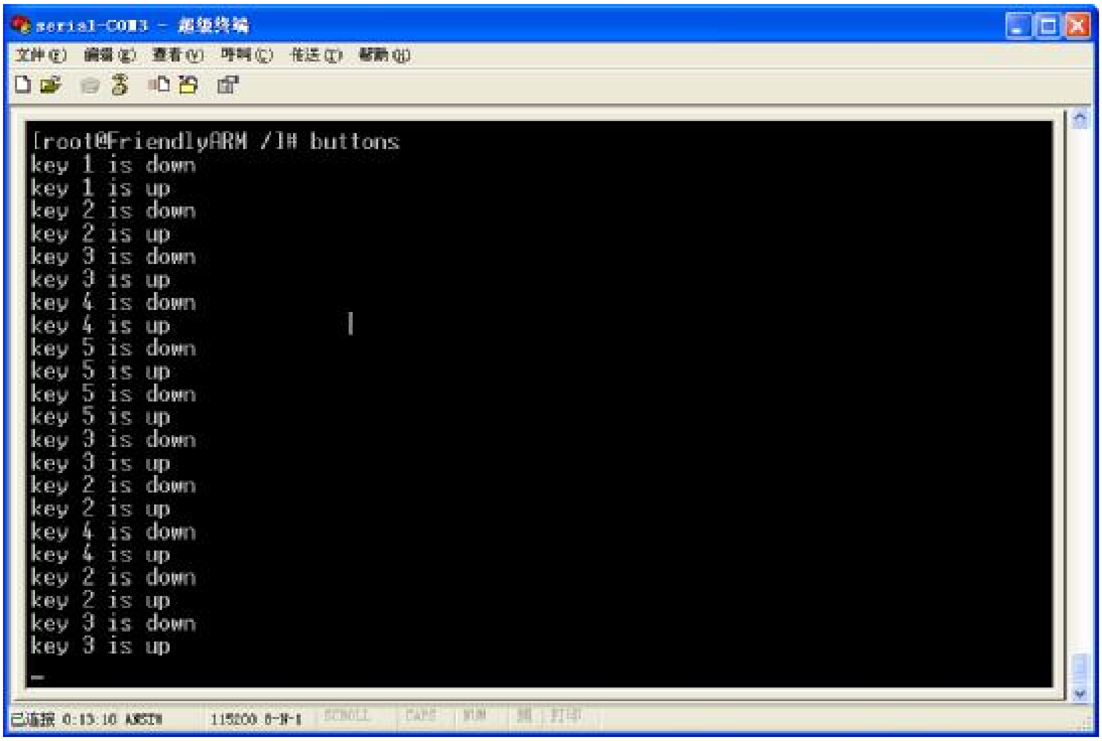
源代码在光盘中的位置：Linux 源代码和工具\examples-20100108.tar.gz

交叉编译器：arm-linux-gcc-4.3.2 with EABI

开发板上对应的设备名：/dev/buttons

对应的内核驱动源代码：Linux-2.6.32.2/drivers/char/GQ2440_buttons.c

在命令行输入“buttons”命令，然后按开发板上的按键，可以显示对应的键值，如图：

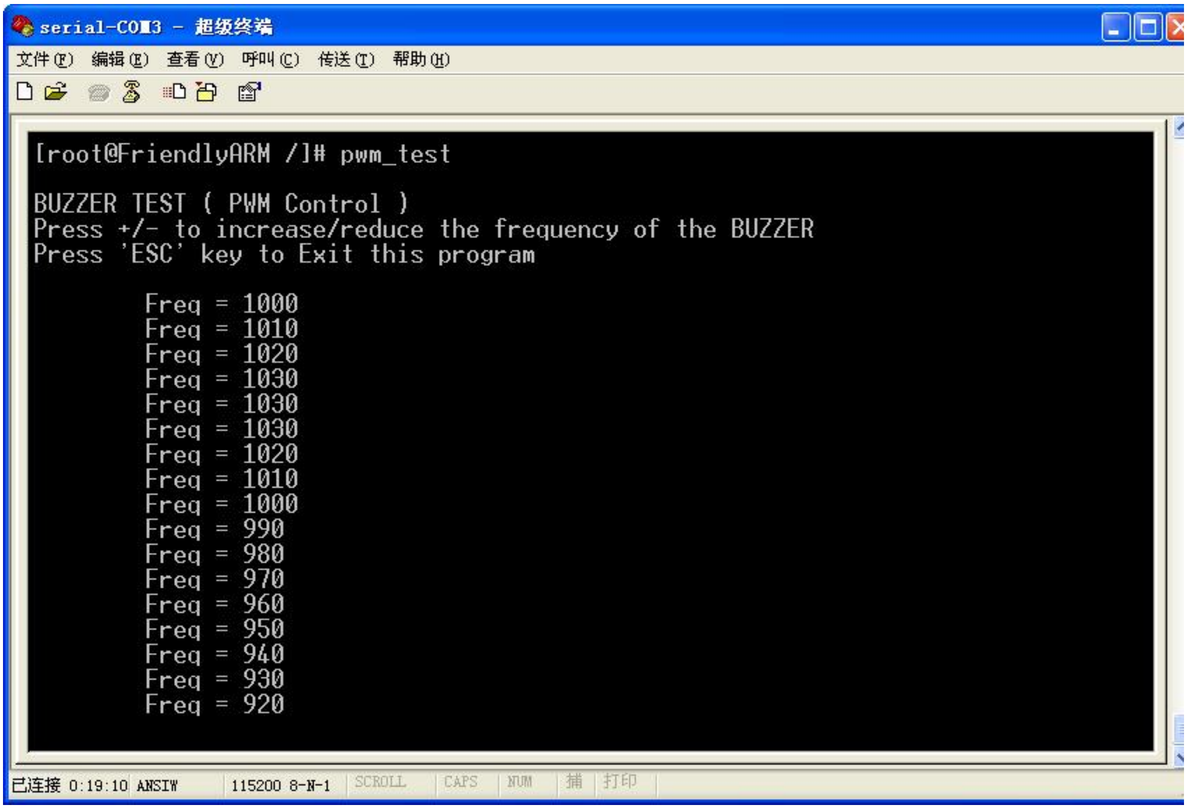


2.5.8 测试蜂鸣器

测试程序名称：pwm_tset
源代码包的名称：Pwm_test.c
源代码在光盘中的位置：Linux 源代码和工具\examples-20100108.tar.gz
交叉编译器：arm-linux-gcc-4.3.2 with EABI
开发板上对应的设备名：/dev/pwm
对应的内核驱动源代码：Linux-2.6.32.2/drivers/char/pwm.c

在命令行种输入：pwm_test

可以听到蜂鸣器的发出的声音，按“+”或者“-”可以改变输出的频率，如图。



按 ESC 键中止该测试。

2.5.9 控制 LCD 的背光

提示:

LCD 背光设备文件：/dev/backlight

在命令行种输入：echo 0 > /dev/backlight 可以关闭 LCD 背光。

在命令行种输入：echo 1 > /dev/backlight 可以打开 LCD 背光。

2.5.10 测试 I2C-EEPROM

测试程序名称：i2c

源代码包的名称：Eeprom.c 24cXX.c

源代码在光盘中的位置：Linux 源代码和工具\examples-20100108.tar.gz

交叉编译器：arm-linux-gcc-4.3.2 with EABI

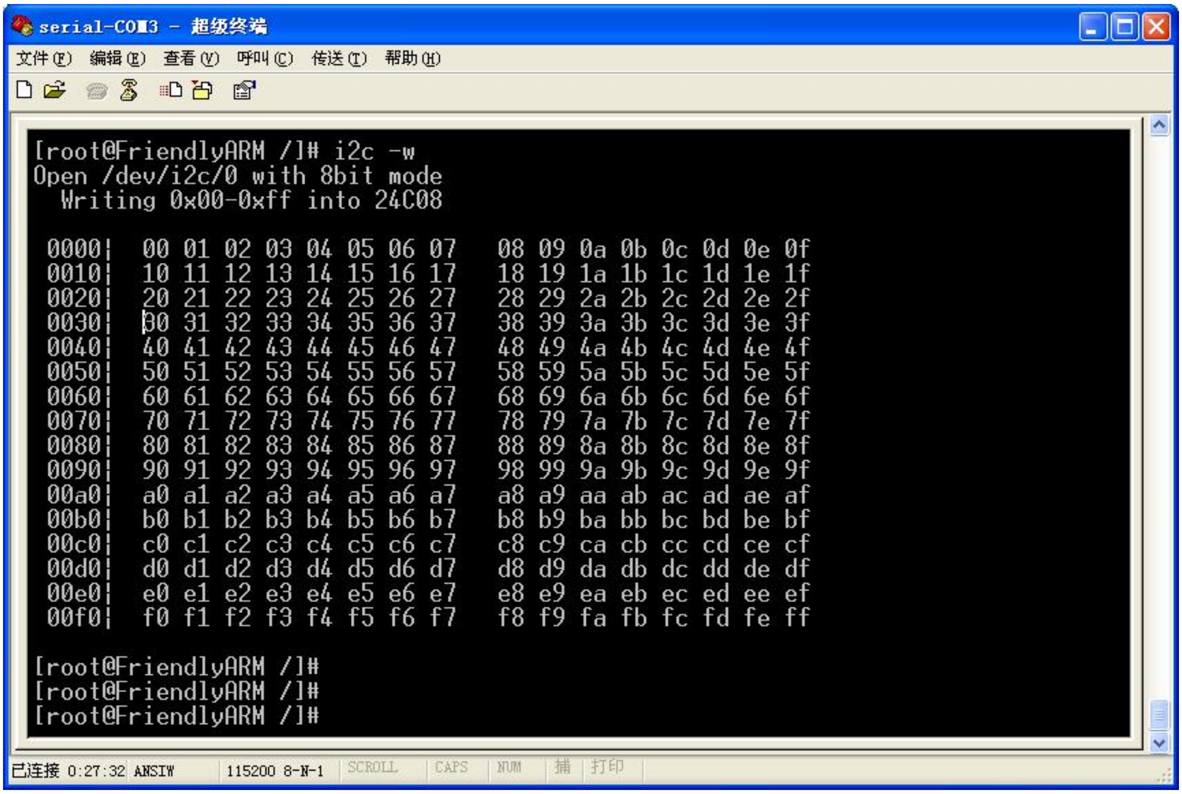
开发板上对应的设备名：/dev/i2c/0

对应的内核驱动源代码：Linux-2.6.32.2/drivers/char/i2c/busses/i2c-s3c2440.c

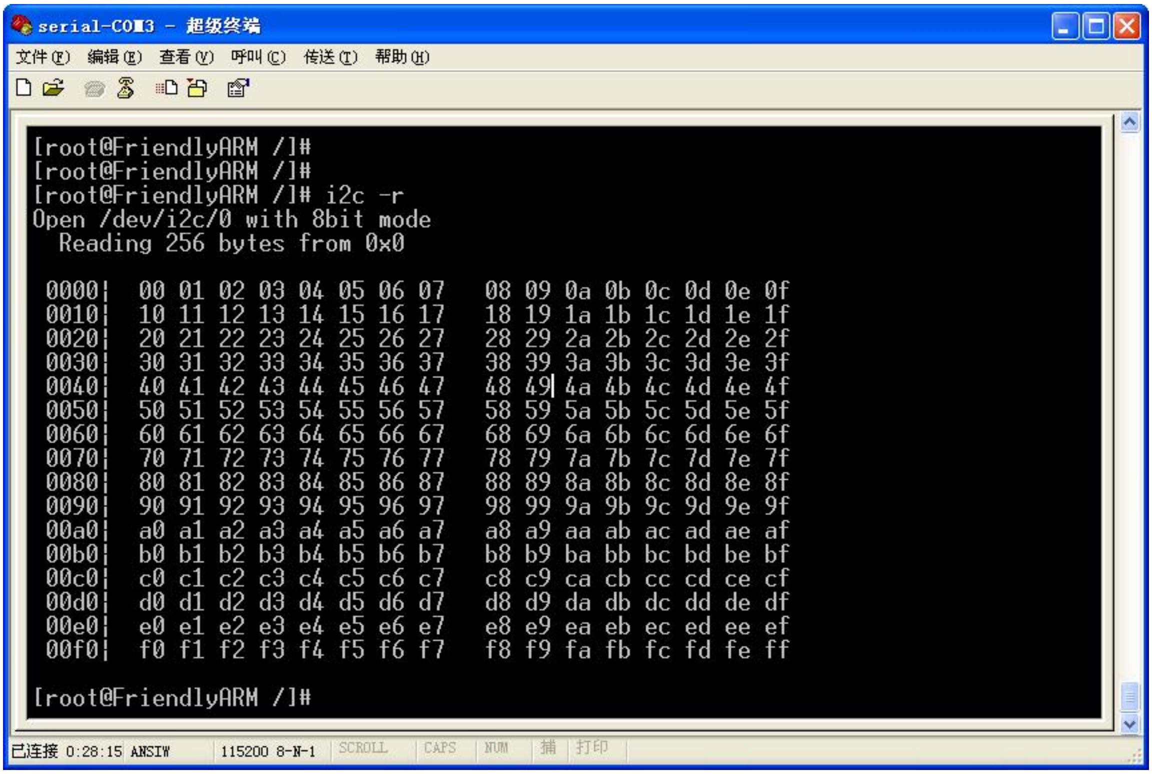
提示：

I2C-EEPROM 的设备文件名：/dev/i2c/0

在命令行种输入：i2c –w 可以向板子的 24C08 器件中写入数据（0x00-0xff）



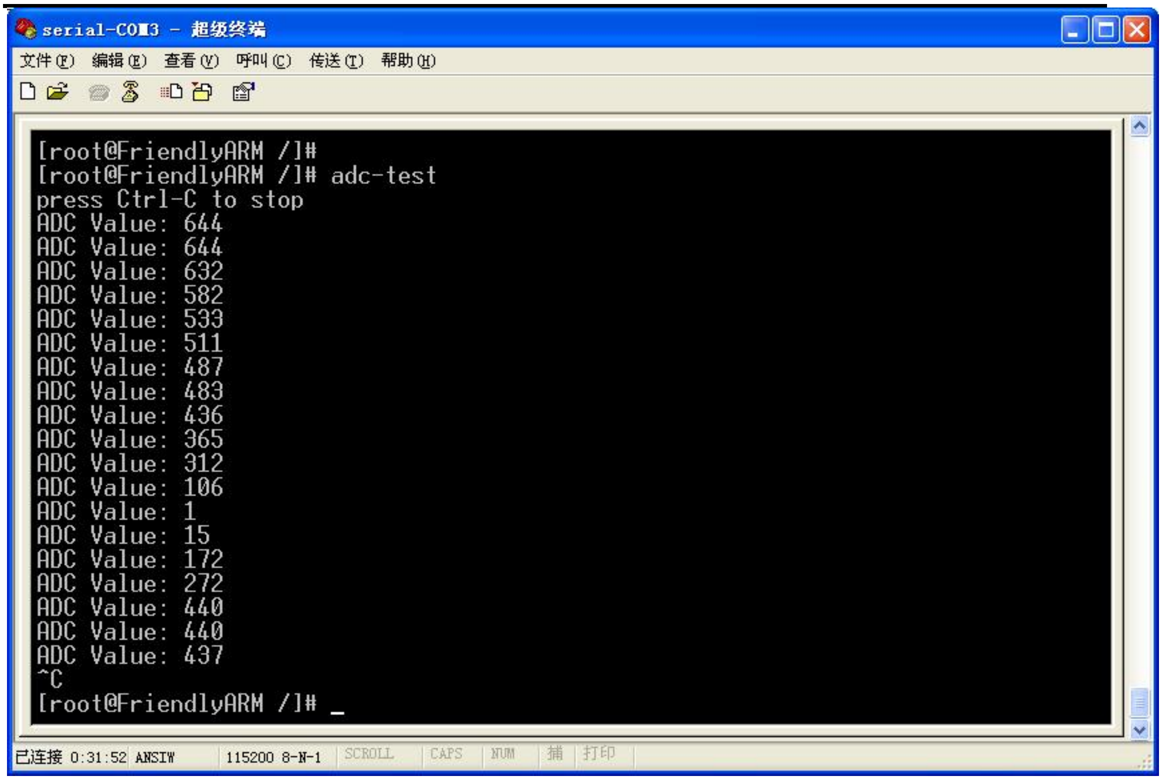
在命令行中输入：i2c –r 可以从板子的 24C08 器件中读出输出：



2.5.11 AD 转换测试

测试程序名称：adc-test
源代码包的名称：Adc-test.c
源代码在光盘中的位置：Linux 源代码和工具\examples-20100108.tar.gz
交叉编译器：arm-linux-gcc-4.3.2 with EABI
开发板上对应的设备名：/dev/adc
对应的内核驱动源代码：Linux-2.6.32.2/drivers/char/GQ2440_adc.c

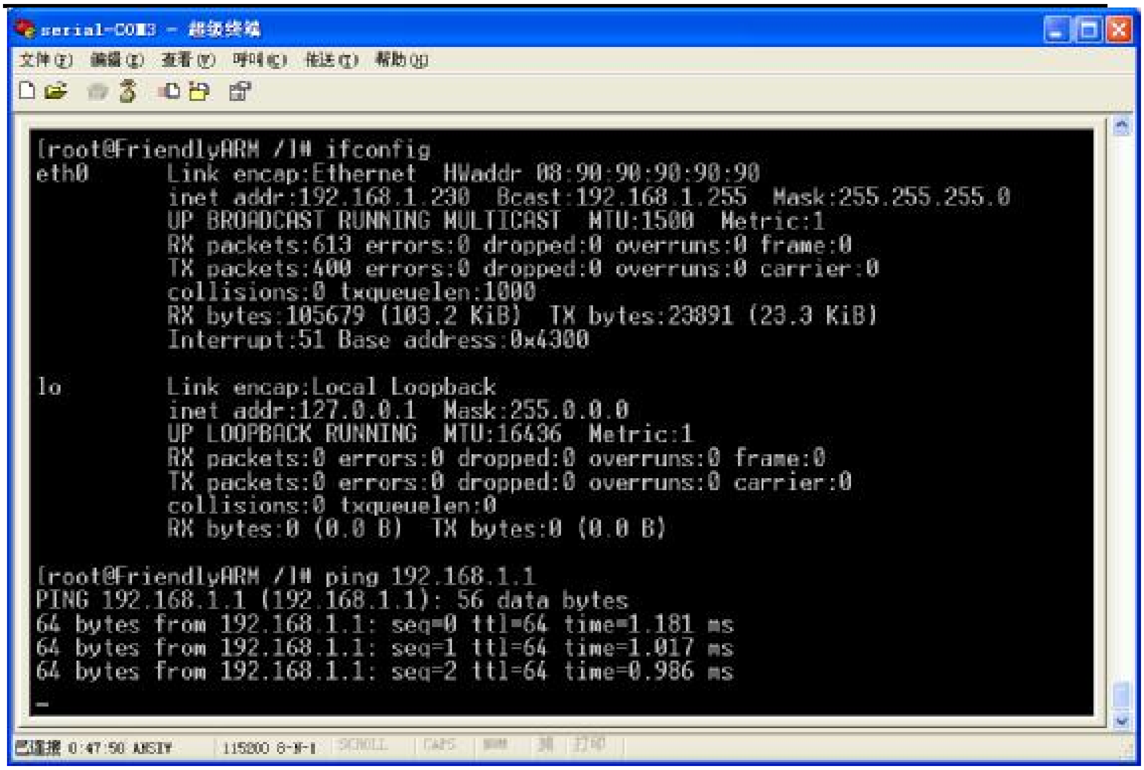
在命令行输入 adc-test 命令，可以进行 ADC 转换测试，调节开发板板上的可调电阻 W1，可以看到从串口终端输出的转换结果。



2.5.12 使用 telnet 上 bbs

telnet 是一个经常被使用的远程登录工具，使用 telnet 功能，可以从开发板登录到其他提供了 telnet 服务器的主机，如果您接入开发板的网络可以上互联网，则可以通过 telnet 命令登录外部的 bbs。

首先，确认开发板的 IP 地址是否为 192.168.1.230，并且是否和局域网内其他主机相通，如图为成功的信息。

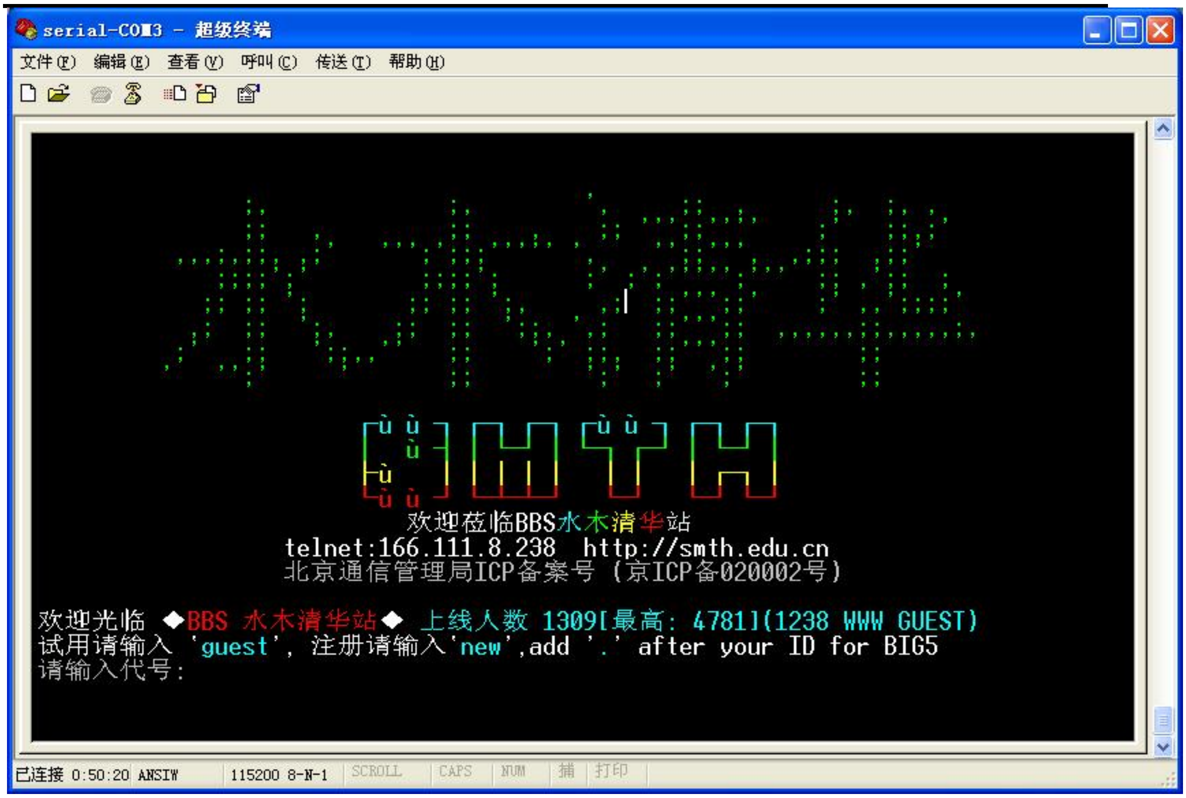


然后设置路由 IP:

#route add default gw 192.168.1.1

最后使用 telnet 命令登录您要登录的主机，在此登录的是水木清华的 bbs。

#telnet 166.111.8.238



2.5.13 如何设置网络以访问互联网网址

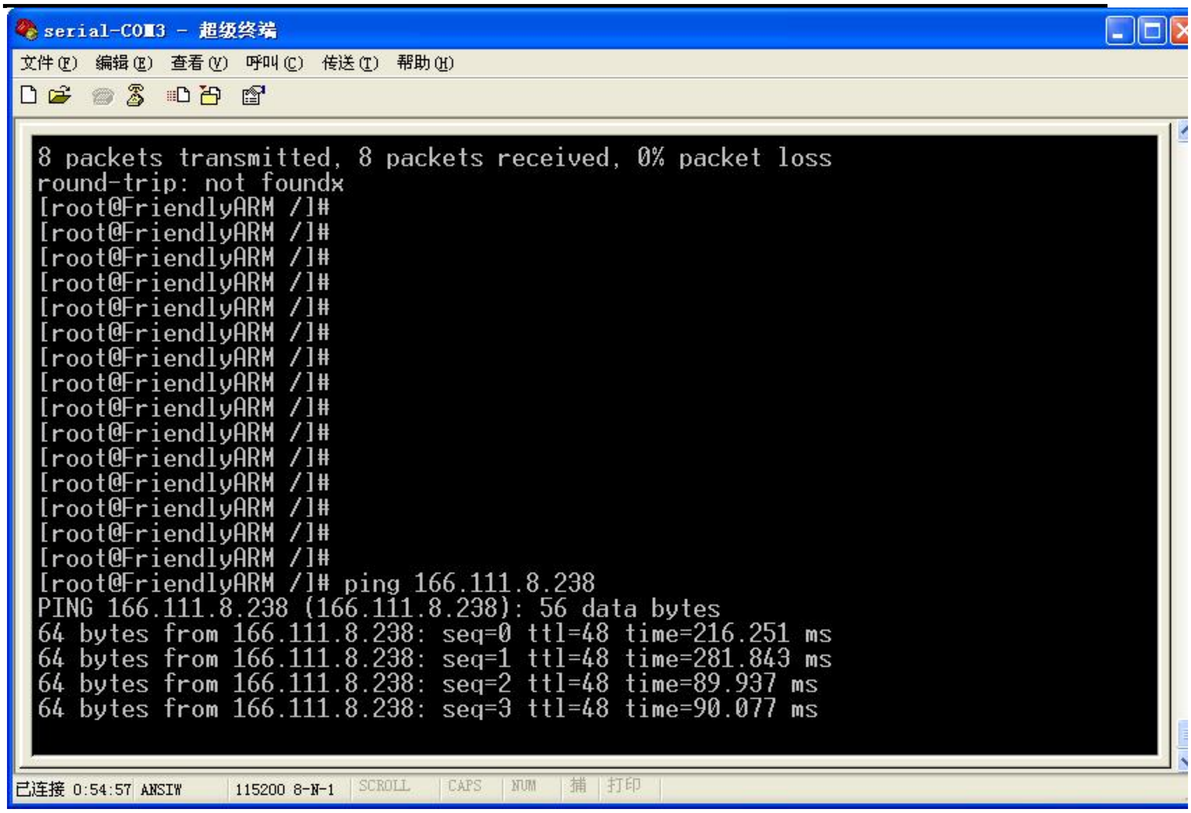
首先要确保你的网络环境可以正常登陆互联网，请记下你的网络环境所使用的网关 IP 地址，比如在我这里是 192.168.1.1，然后使用 route 进行设置:

route add default gw 192.168.1.1

这时你就可以直接访问互联网上的数字 IP 地址了，比如 ping 一下华南木棉的 BBS(其 IP 地址为 166.111.8.238):

#ping 166.111.8.238

如图所示表示可以 ping 通外面的网络:



要能 ping 通外部网络的实名网址，还需要设置好域名解析服务器，先查看一下您当前网络所使用的 DNS 服务器 IP 地址(可以询问您的网络管理员):



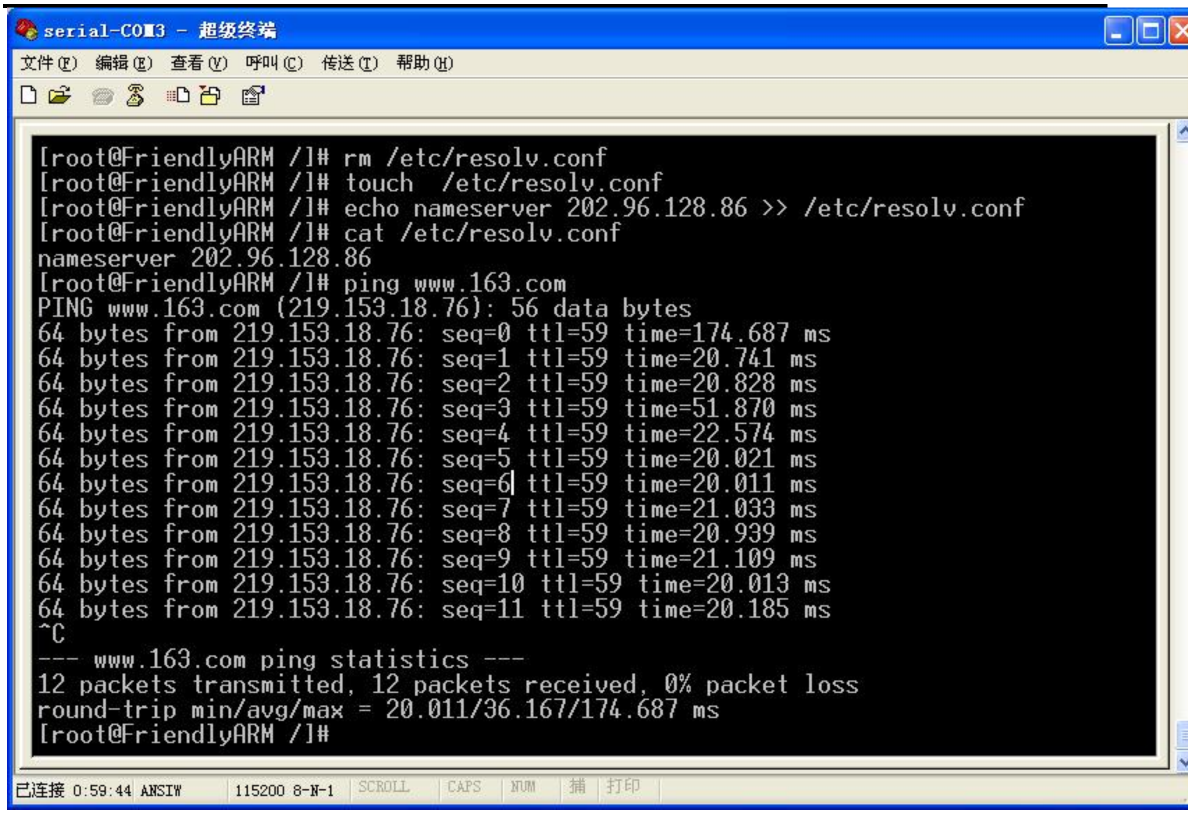
比如，我这里 DNS 服务器的 IP 为“202.96.128.86”，则在开发板中这样设置:

#rm /etc/resolv.conf *;首先删除以前的配置文件*

#touch /etc/resolv.conf *;重新生成一个 resolv.conf 文件*

#echo nameserver 202.96.128.86 >> /etc/resolv.conf *;使用实际的 DNS 服务器 IP 配置 resolv.conf 文件*

可以这里主要是修改/etc/resolv.conf 文件，当然你也可以直接使用 vi 进行修改。全部过程如下图所示:

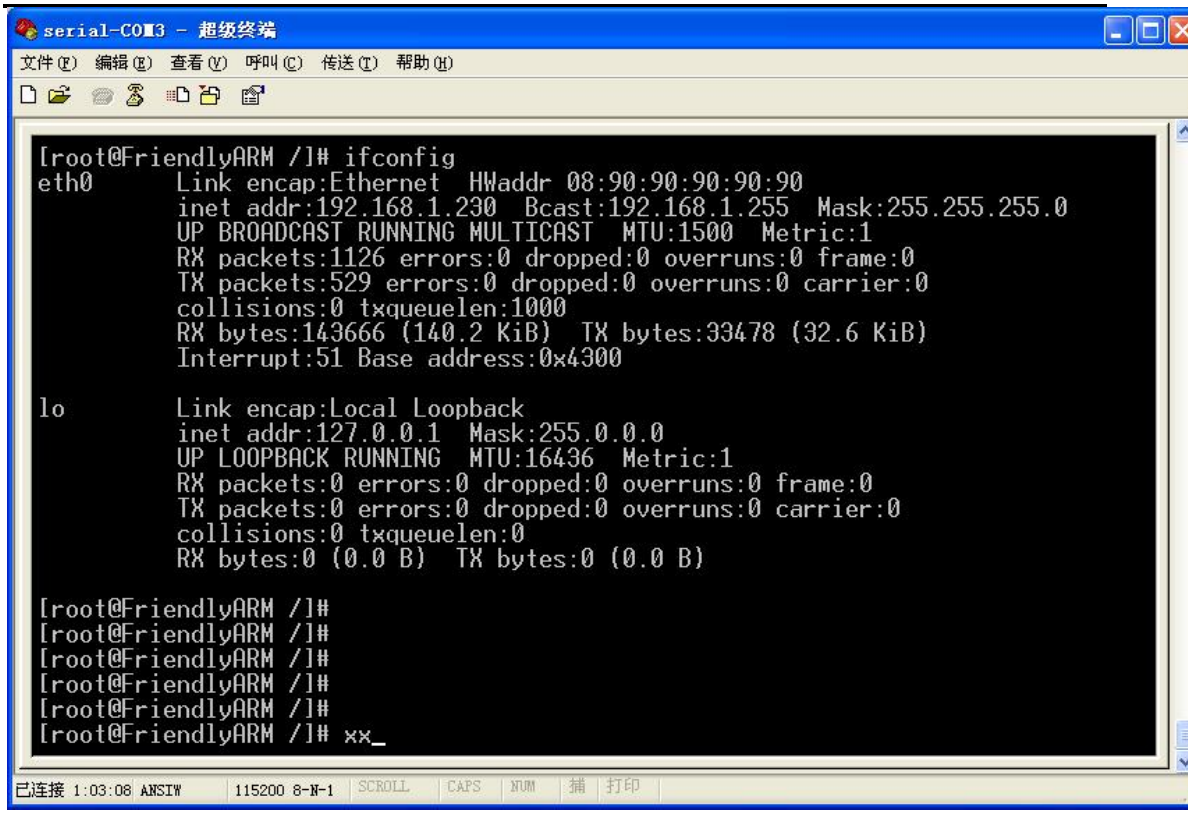


2.5.14 如何设置 MAC 地址

开发板中所使用的 MAC 地址是"软"性的，因此你可以通过 ifconfig 命令对它进行重置，以适应于在同一个网络环境中使用多片开发板的情况，具体操作如下：

首先使用 ifconfig 查看一下当前的 mac 地址，运行：

#ifconfig　　　　　*;注意后面不要跟任何内容*



可以看到当前的 mac 地址为"08：90：90：90：90：90"，这是在网卡驱动中默认的 mac 地址，它已经被写死到内核中，除非更改网卡的驱动源代码并重新编译得到新内核。要在运行的系统中动态更改 mac 地址，先关闭当前网络，并使用 ifconfig 重置 mac 地址：

#ifconfig eth0 down

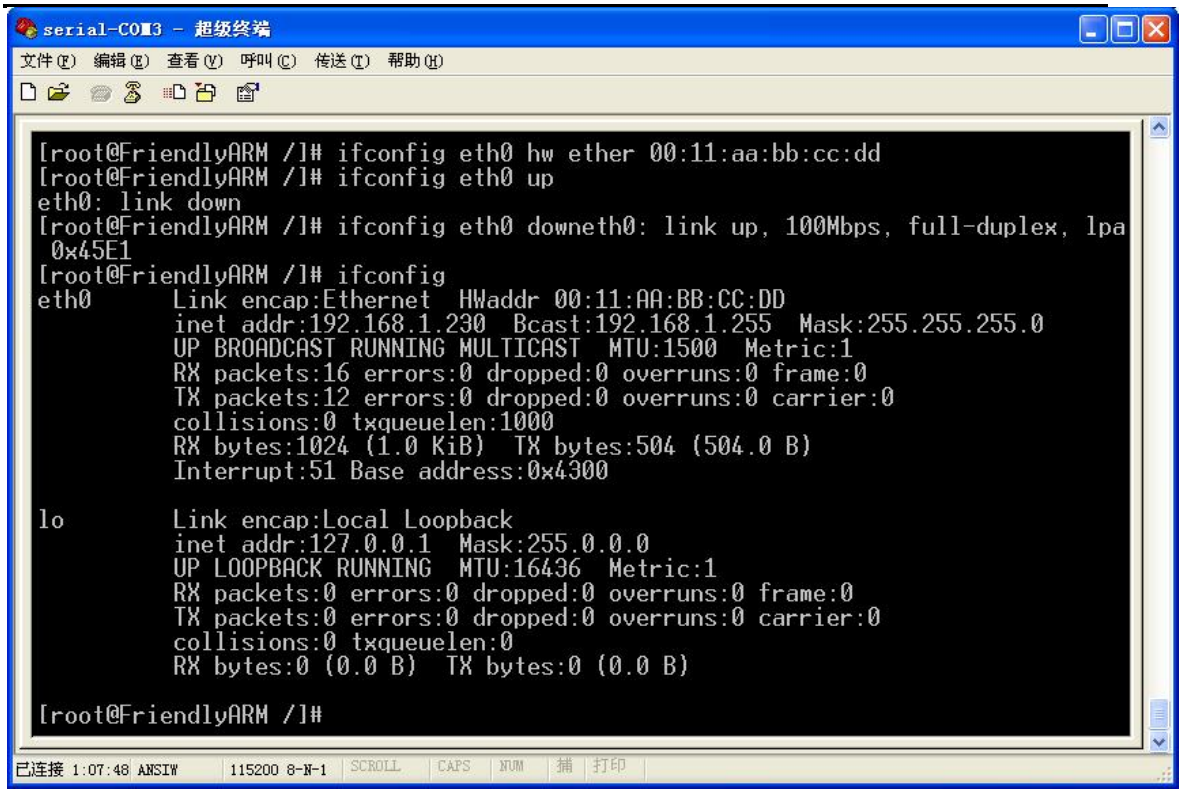
#ifconfig eth0 hw ether 00:11:AA:BB:CC:DD *;提示：a,b,c,d,e,f 可以为小写*

再开启网络，并使用 ifconfig 查看设置以后的 mac 地址，使用 ping 检验网络是否依然可通：

#ifconfig eth0 up

#ifconfig

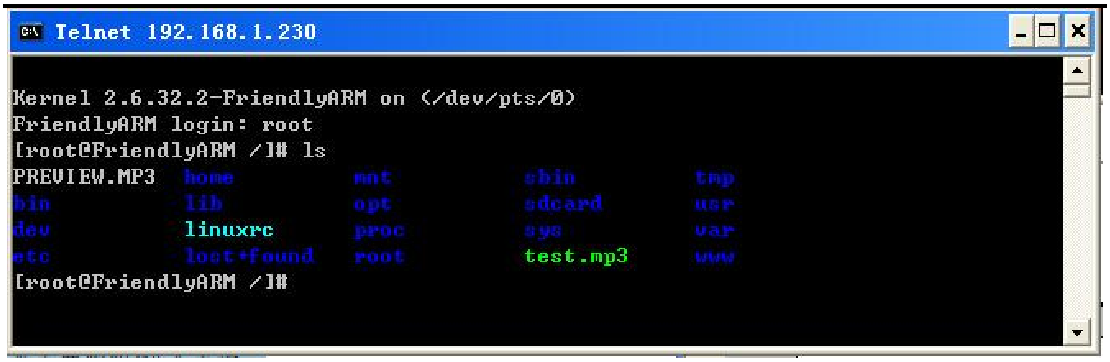
#ping 192.168.1.1



2.5.15 如何使用 Telnet 远程登录开发板

开发板开机正常运行后，其实已经启动了一个 Telnet 服务，因此用户也可以通过网络远程登录开发板。

在 Windows 的命令行窗口输入“telnet 192.168.1.230”，如图出现登录界面，输入“root”(不需要密码)进入系统



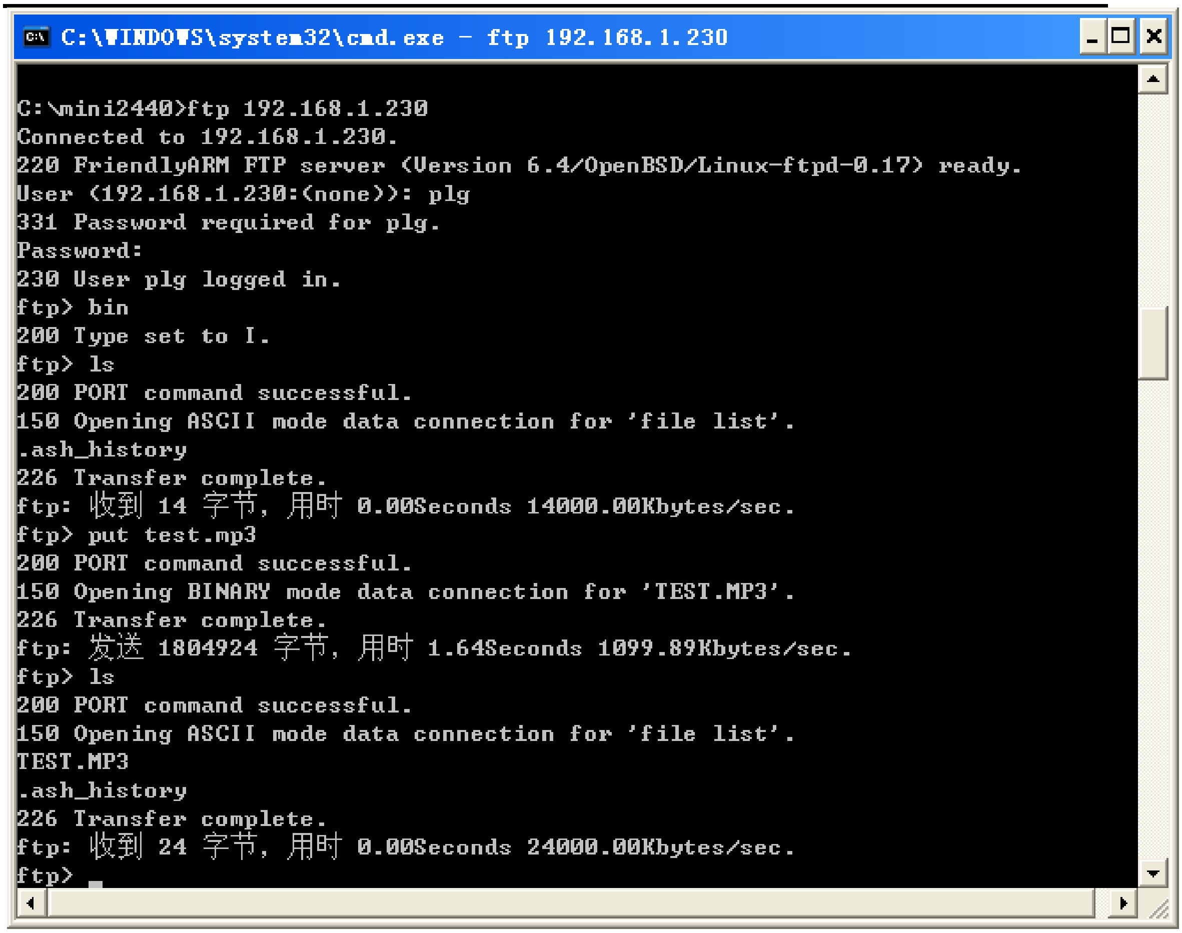
2.5.16 使用 ftp 传递文件

无论在 Linux 系统还是 windows 系统中，一般安装后都自带一个命令行的 ftp 命令程序，使用 ftp 可以登录远程的主机，并传递文件，这需要主机提供 ftp 服务和相应的权限；本开发板不仅带有 ftp 命令，还在开机时启动了 ftp 服务。为了方便测试，我们可以从 PC 机的命令行窗口登录开发板，并向开发板传递文件。

注意：请确保您执行 ftp 所在的目录有需要上传的文件，这里是 test.mp3

说明：登录开发板的 ftp 帐号为：plg 密码为：plg

传送完毕，您可以在串口终端看到目标板的/home/plg 目录下多了一个 test.mp3 文件。



2.5.17 通过网页控制板上的 LED

在 web server 测试页面中点“网络控制 LED 测试”项，会出现 LED 测试控制页面，如图：



您可以使用网页中的各个测试项目进行测试，其中的“LED 测试”将会通过 CGI 程序来控制板上的 LED 灯，其中包括 2 种方式的显示类型和三种不同的显示速度。

如果要停止 web 服务器，则在命令提示符下输入以下命令:

#/etc/rc.d/init.d/httpd stop

要重新启动则输入:

#/etc/rc.d/init.d/httpd start

2.5.18 如何挂接使用网络文件系统 NFS

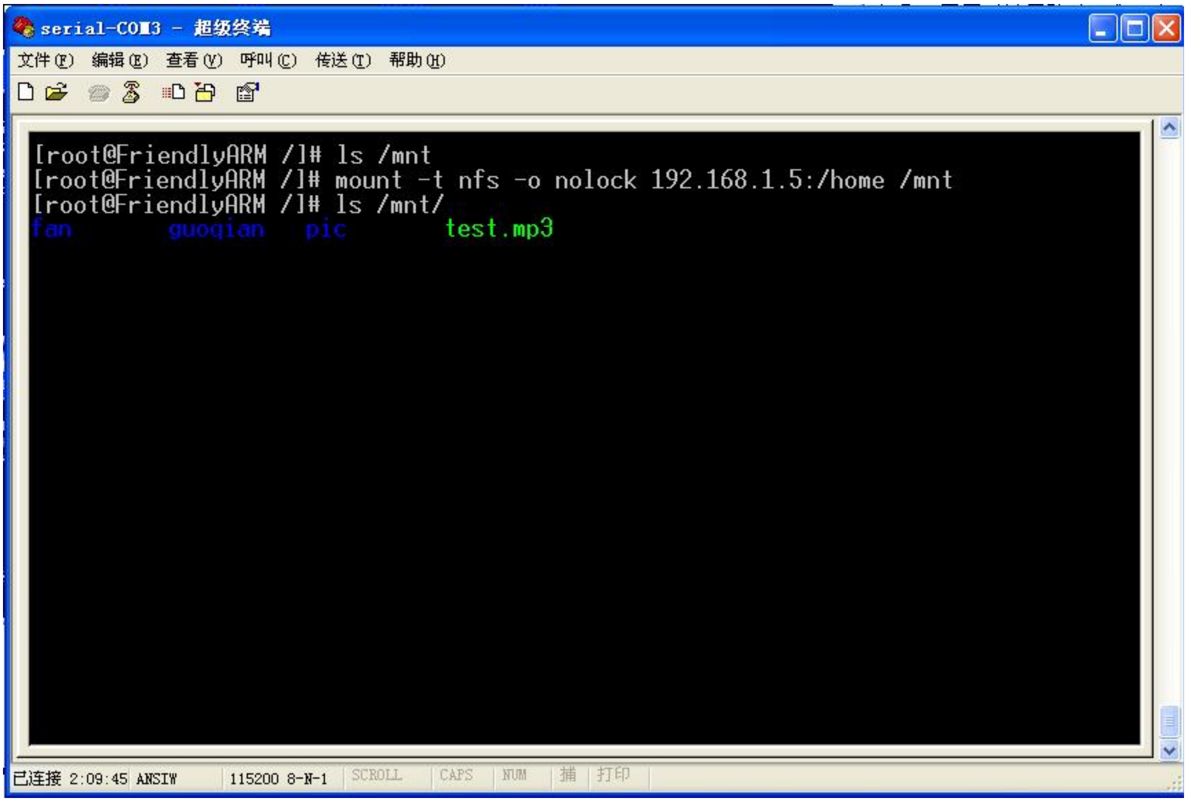
在进行该测试之前，请先按照 5.5 一节搭建好 NFS 服务器系统，然后在命令行输入以下命令（假定服务器的 IP 地址为 192.168.1.5）：

#mount –t nfs –o nolock 192.168.1.5:/home /mnt

挂接成功，您就可以进入/mnt 目录进行操作了，如下图所示。

取消挂接的命令如下:

#umount /mnt



2.5.19 使用 USB 无线网卡

Linux-2.6.32.2 内核已经自带了大部分 USB 无线网卡的驱动，其中包括目前最流行、

性价比最高的 TP-Link USB WiFi 无线网卡：TL-WN321G+，本开发板的缺省内核映象已经加入了本模块的支持。

提示：本手册的 6.3 章节有关于在内核中配置无线网卡驱动的说明

把 USB 无线网卡插入开发板，会出现如下信息：

```
[root@FriendlyARM /]# usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: New USB device found, idVendor=148f, idProduct=2573
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: 54M.USB.......
usb 1-1: Manufacturer: Ralink
usb 1-1: configuration #1 chosen from 1 choice
wmaster0 (rt73usb): not using net_device_ops yet
wlan0 (rt73usb): not using net_device_ops yet

[root@FriendlyARM /]#
```

这说明已经识别到该无线网卡，然后通过步骤命令开始配置。

(1)首先关闭开发板的有线网卡 DM9000

```
[root@FriendlyARM /]# ifconfig eth0 down
```

(2)加载 USB WiFi 无线网卡

```
[root@FriendlyARM /]# ifconfig wlan0 up
rt73usb 1-1:1.0: firmware: requesting rt73.bin
[root@FriendlyARM /]#
```

(3)扫描可用的无线网络

```
[root@FriendlyARM /]# iwlist scanning | grep ESSID
lo Interface doesn't support scanning.
eth0 Interface doesn't support scanning.
wmaster0 Interface doesn't support scanning.
ESSID:"FRIENDLY-ARM"
ESSID:"NETGEAR"
ESSID:"TP-LINK"
```

```
[root@FriendlyARM /]#
```

(4)选择要连接的无线网络

```
[root@FriendlyARM /]# iwconfig wlan0 essid "FRIENDLY-ARM"
```

(5)输入该网络的安全密码

```
[root@FriendlyARM /]# iwconfig wlan0 key s:12345
```

(6)连接到指定的 AP(无线路由)

```
[root@FriendlyARM /]# iwconfig wlan0 ap auto
```

(7)设置无线网卡的 IP 地址

```
[root@FriendlyARM /]# ifconfig wlan0 192.168.1.120
```

(8)使用 ping 命令检测无线网连通状况

```
[root@FriendlyARM /]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=42.804 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=5.020 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=5.021 ms
```

以上整个过程如下所示:

```
[root@FriendlyARM /]# usb 1-1: new full speed USB device using s3c2410-ohci and
address 2
usb 1-1: New USB device found, idVendor=148f, idProduct=2573
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: 54M.USB.......
usb 1-1: Manufacturer: Ralink
usb 1-1: configuration #1 chosen from 1 choice
wmaster0 (rt73usb): not using net_device_ops yet
wlan0 (rt73usb): not using net_device_ops yet
[root@FriendlyARM /]# ifconfig eth0 down
[root@FriendlyARM /]# ifconfig wlan0 up
rt73usb 1-1:1.0: firmware: requesting rt73.bin
[root@FriendlyARM /]# iwconfig wlan0 key s:12345
[root@FriendlyARM /]# iwconfig wlan0 essid "FRIENDLY-ARM"
[root@FriendlyARM /]# iwconfig wlan0 ap auto
[root@FriendlyARM /]# ifconfig wlan0 192.168.1.120
```

```
[root@FriendlyARM /]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=42.804 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=5.020 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=5.021 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 5.020/17.615/42.804 ms
[root@FriendlyARM /]#
```

2.5.20 设置并保存系统实时时钟

Linux 中更改时间的方法一般使用 date 命令，为了把 S3C2440 内部带的时钟与 linux 系统时钟同步，一般使用 hwclock 命令，下面是它们的使用方法:

(1) **date -s 012014202011** #设置时间为 2011-01-20 14:20

(2) hwclock -w #把刚刚设置的时间存入 S3C2440 内部的 RTC

(3).开机时使用 hwclock -s 命令可以恢复 linux 系统时钟为 RTC，一般把该语句放入 /etc/init.d/rcS 文件自动执行。

注意：**我们提供的系统已经把 hwclock –s 命令写入 rcS 文件。**

2.5.21 如何掉电保存数据到 Flash

由于本系统采用了可读写文件系统 yaffs(在嵌入式系统中，专门管理 Flash 存储器的一种文件系统)，因此可以很方便的动态保存数据，掉电后不会丢失。

2.5.22 如何设置开机自动运行程序

借助启动脚本可以设置各种程序开机后自动运行，也可以设置其他系统设置，这有点

类似于 Windows 系统中的 Autobat 自动批处理文件，启动脚本的位于板子的 /etc/init.d/rcS，内容如下(实际内容可能与此不完全一致):

```
#! /bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel
#
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#
trap ":" INT QUIT TSTP
/bin/hostname FriendlyARM
/bin/mount -n -t proc none /proc
/bin/mount -n -t sysfs none /sys
/bin/mount -n -t usbfs none /proc/bus/usb
/bin/mount -t ramfs none /dev
echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp
```

```
/sbin/hwclock -s
syslogd
/etc/rc.d/init.d/netd start
echo " " > /dev/tty1
echo "Starting networking..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/httpd start
echo " " > /dev/tty1
echo "Starting web server..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/leds start
echo " " > /dev/tty1
echo "Starting leds service..." > /dev/tty1
echo " "
sleep 1
/sbin/ifconfig lo 127.0.0.1
/etc/init.d/ifconfig-eth0   ;设置开机静态 IP 地址，这是一个脚本，你可以使用 vi 打开并编辑
#/bin/qtopia &
echo " " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
```

2.5.23 如何使用命令进行屏幕截图

使用 snapshot 命令可以对当前的 LCD 显示进行截图，并保存为 png 格式的图片。

#snapshot pic.png

执行该命令将把当前 LCD 显示进行抓图，并保存为 pic.png 文件。



2.6 体验 WindowsCE 5.0

WindowsCE5 的烧写文件位于光盘"\images\wince5.0"目录中。

请按照"安装和更新系统程序"一章下载烧写您所需要的 wince 映象文件(此示例中

烧写的是 NK_T35.bin)。安装完毕，请把开发板的 S2 开关设置为 Nand Flash 启动系统。如果你在安装时选择的 bootloader 是 Nboot，开机启动系统你将会看到一个开机图片，

然后会有进度条指示系统正在装载中，装载完毕即可出现 WindowsCE 的启动画面，然后进入桌面系统，如图：



启动画面



wince 桌面

2.6.1 触摸屏校正

触摸屏驱动程序源代码位于 BSP 的如下目录:

GQ2440\WinCE 源代码和软件\WindowsCE5.0\GQ2440_10_06_BSP\Src\Drivers\Touch

缺省安装的 wince 系统的触摸屏校正参数一般适用于三星 3.5"LCD，但因为每个触摸

屏的物理特性不同，有时可能不太准确，特别是不同尺寸的时候，这时就需要重新校正，如下步骤。

请先接上 USB 鼠标，点【开始】|【设置】|【控制面板】，找到“笔针”图标，双击打开“笔针属性”窗口，点“校准”选项卡的“再校准”按钮，开始重新校正。



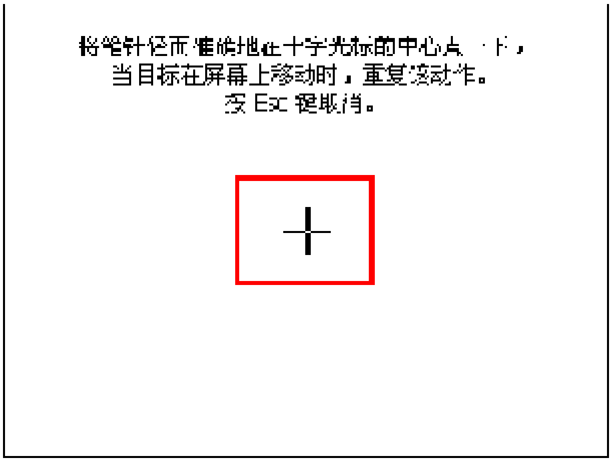
找到“笔针”图标，双击打开它：

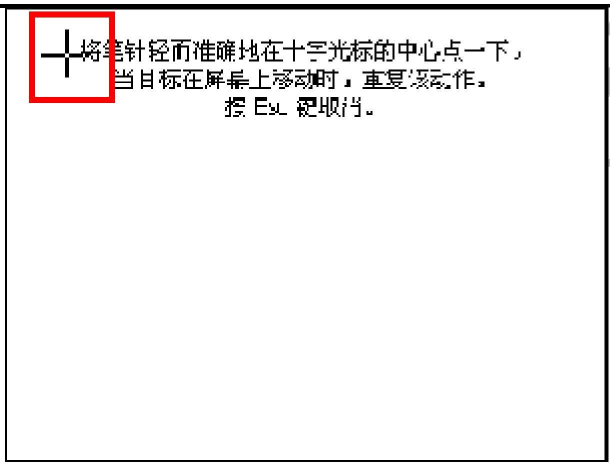


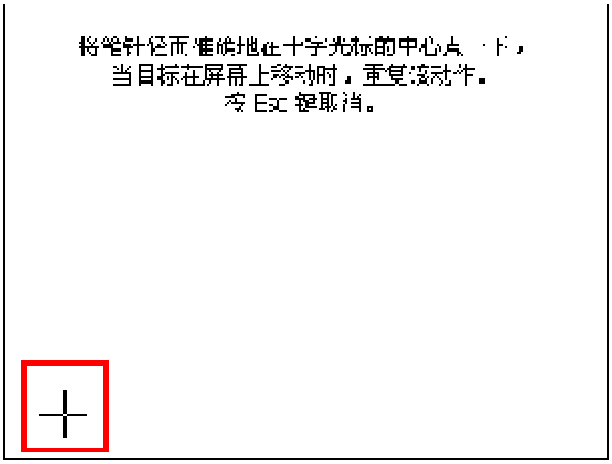
然后进入“笔针属性”，点击“校准”进入校正选项页面：

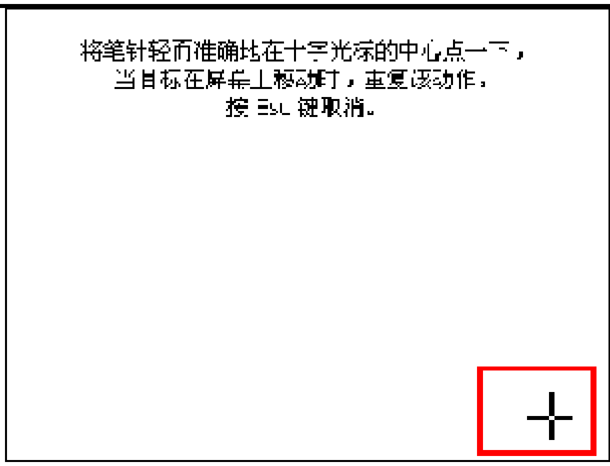


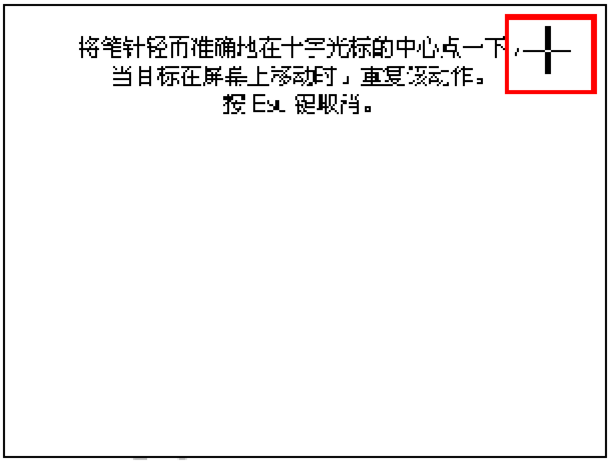
点击"再校准"选项，就可以开始校正了，总共要校正五个点，下面的连续五个图分别是各个点的截图，红色框中的十字架中心就是要校正的点，触摸板点击请尽量对准，然后轻触 1s 左右。



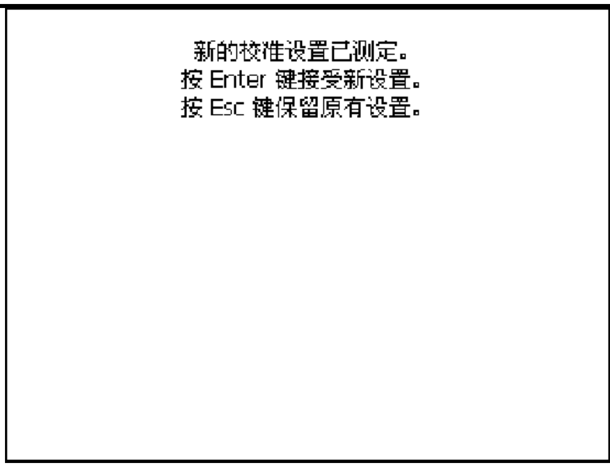

将笔针轻而准确地在十字光标的中心点一下，
当目标在屏幕上移动时，重复该动作。
按 Esc 键取消。


将笔针轻而准确地在十字光标的中心点一下，
当目标在屏幕上移动时，重复该动作。
按 Esc 键取消。





完成上面五点校正之后，出现下图所示，然后点击任意地方继续：



返回下图所示界面，然后点击“OK”选项，即可完成校正：



如果想保存本次校准的参数，通过【开始】|【挂起】，然后重启就可以了。

2.6.2 设置网络参数以连接互联网

网卡驱动程序源代码位于 BSP 的如下目录：

GQ2440\WinCE 源代码和软件\WindowsCE5.0\GQ2440_10_06_BSP\Src\Drivers\dm9000

只有正确设置了 IP 地址和网关以及 DNS 等参数，才可以使用开发板连接使用互联

网或者局域网。

设置网络参数的步骤和标准的 Windows 系统十分相似，点【开始】|【控制面板】，找到网络设置选项，并找到相应的网卡 DM9CE1 如图。





双击打开 DM9CE1 图标，如图，在这里，你可以设置静态的 IP 地址，也可以设置动态获取 IP 的方式，按照你所在的网络环境实际参数填写即可，如图为缺省的设置参数。





确定网络设置参数无误，就可以打开浏览器上互联网了。



在地址栏输入网址:



浏览网页:





2.6.3 背光设置

网卡驱动程序源代码位于 BSP 的如下目录：

GQ2440\WinCE 源代码和软件\WindowsCE5.0\GQ2440_10_06_BSP\Src\Drivers\Backlight

在此，你可以设置背光常亮，或者延时关闭，通过触摸屏、鼠标、板上的按键等都可以唤醒背光，实际上，此处实现的背光设置已经属于电源管理的一部分，只不过本开发板仅有外部电源支援。

要进入背光设置界面，有 2 种方式:
可以在空白的桌面处通过点击鼠标右键，再点属性
也可以点【开始】|【控制面板】|【显示】



如图为背光设置界面的缺省设置



可见，使用外部电源供电时，如果在 30 秒之内触摸屏或者按键等 HMI 设备没有动作，

背光将会自动熄灭，你也可以把时间改为其他数值，设置完毕，只需耐心等待即可看到结果了。

为了让背光常亮，可以把上面界面中“使用外部电源并且设备空闲”取消就可以了。

2.6.4 设置实时时钟并保存

网卡驱动程序源代码位于 BSP 的如下目录:

GQ2440\WinCE 源代码和软件\WindowsCE5.0\GQ2440_10_06_BSP\Src\Drivers\Rtc

点任务栏右下角的时间或者通过点击【控制面板】中的“日期/时间”图标，出现时间设置窗口，根据提示进行设置就可以了，设置完毕，点“OK”退出，设置时间不需要点“挂起”保存。





2.6.5 使用 U 盘

在 wince 中使用优盘和使用标准的 windows 使用 U 盘类似，当 WINCE 系统启动以后，把优盘插入 USB Host 接口，这时板子给优盘供电，优盘的指示灯会闪烁，等待几秒系统就自动加载优盘了，这时可以双击桌面的“我的电脑”图标，打开资源管理器，可以看到优盘的盘符：【硬盘】，双击硬盘即可进入优盘进行数据读写了。如下图所示。





2.6.6 使用 SD/MMC 卡

说明：因为采用了最新架构的 BSP，目前开发板最大可以支持 32G 普通和高速 SD 卡。　　把 SD/MMC 卡插入板上的 SD 插槽，资源管理器中就可以看到 SD 卡的盘符：Storage Card，双击打开进入该目录，就可以对 SD/MMC 卡进行读写了。



复制 SD 卡中的一个文件：



2.6.7 保存数据到 Flash 掉电不丢失

我们提供的 wince 的 BSP 包支持把 Flash 上剩余空间作为电子硬盘使用，可以实现掉电保存数据到 Flash，它在 wince 里面以【Resident Flash】盘符的形式存在。

下面测试该功能：

双击打开【我的设备】图标，可以看到名为【Resident Flash】的盘符：



双击它进入，就可以进行数据的读写操作了：



这里复制 U 盘里面发的 CECap（这是一个可用于 wince 截屏的软件）文件到该目录。



然后重启板子后可以看到文件还是存在:



2.6.8 使用文档编辑器

使用系统自带的文档编辑器 WordPad 可以进行文档编辑，如图，点击桌面图标【Mirosoft WordPad】打开编辑器，可以使用软键盘进行简单的文档编辑:







2.6.9 浏览图片和播放音频、视频

使用 wince 自带的浏览器功能和 Media Player 可以在 wince 上进行图片浏览和视频播放，也可以借助第三方软件如图片浏览器——“imageviewer.exe”，视频播放器——TCPMP 等。下面是使用这些软件进行图片浏览和音频、视频播放的效果图：



Media Player 播放 mp3:



TCPMP 播放视频:



2.6.10 使用命令行提示符

在 wince 下使用命令行提示符和在 pc 的 windows 下使用命令行提示符方法类似，只需通过【开始】|【运行】，然后输入 cmd，就可以进入到命令行提示符界面，如图：





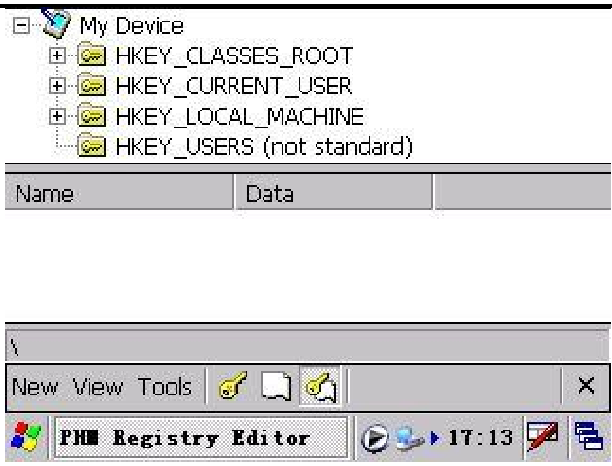
2.6.11 屏幕旋转并保存

说明：屏幕旋转的驱动程序已经包换在 LCD 驱动中，它不需要特别的硬件支持，只需要修改注册表的相应项。

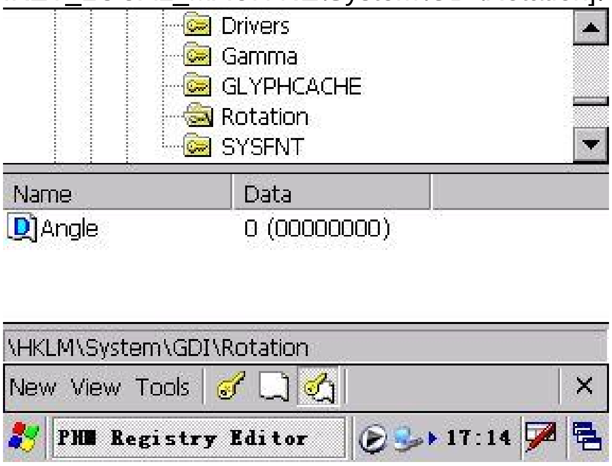
LCD 驱动的位置：GQ2440\Src\Drivers\Display

(1) 通过【开始】|【运行】，然后输入 regedit 打开注册表，如图

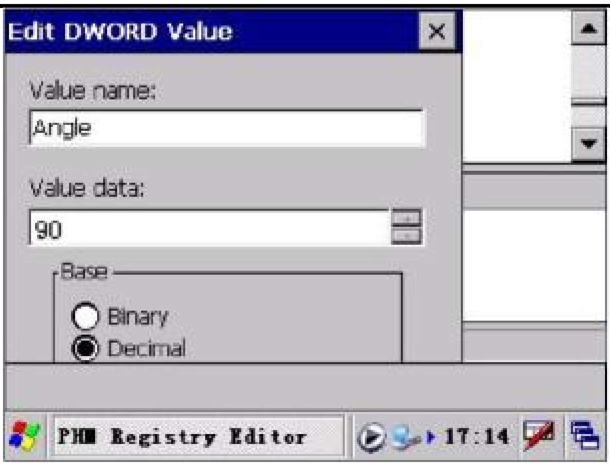




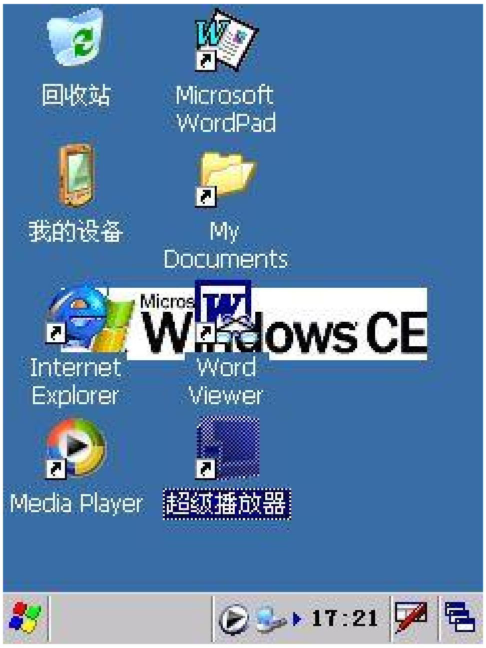
找到注册表项[HKEY_LOCAL_MACHINE\System\GDI\Rotation]:



可以看到 Angle 默认值为 0，代表不选择，我们把这个值设置为 90 度，使屏幕竖起来



重启 wince，可以看到屏幕旋转的效果：



2.6.12 使用 ActiveSync 进行 USB 同步通讯

注意：安装同步连接的 USB 驱动时，请使用光盘中的“\windows 平台工具\CE 用同步 USB 驱动”

我们假定您已经按照第九章的步骤安装好驱动和 ActiveSync 同步软件。

在默认情况下，当 WINCE 系统启动以后，USB Device 连接和设置就已经做好了，这时可以直接用 USB 连接开发板和 PC 机，即可看到如下成功连接的情况:



第3章 系统安装及更新

本章内容主要通过图解介绍如何通过USB 如何安装更新系统。

很多人特别是初学者认为，安装嵌入式系统是十分复杂的事情，需要很高的技术水平，特别是Linux 系统，总以为要输入很多的命令进行配置安装。通过本节的介绍，你将会发现安装和更新系统简直比在 PC 上安装 Windows 系统还要简单，只要几分钟的时间，输入不超过5 个字母，您就可以随意更新您的系统了。

提示：**本节内容的各个小节均具有独立性，用户可以根据自己的目的需要进行独立阅读操作。**

本小节的工作环境为 Windows/2000/XP。

3.1 安装 Linux 系统

注意：**本小节假定您已经按照前面的方法安装了 USB 驱动，并把开发板设置为 NOR Flash 启动，系统更新和安装完毕请设置为 Nand Flash 启动，设置方法请参考前面的章节。**

说明：安装 Linux 所需要的二进制文件位于光盘的 images\linux 目录中。

安装 Linux 系统主要有以下步骤：

(1)对 Nand Flash 进行分区

(2)安装 bootloader

(3)安装内核文件

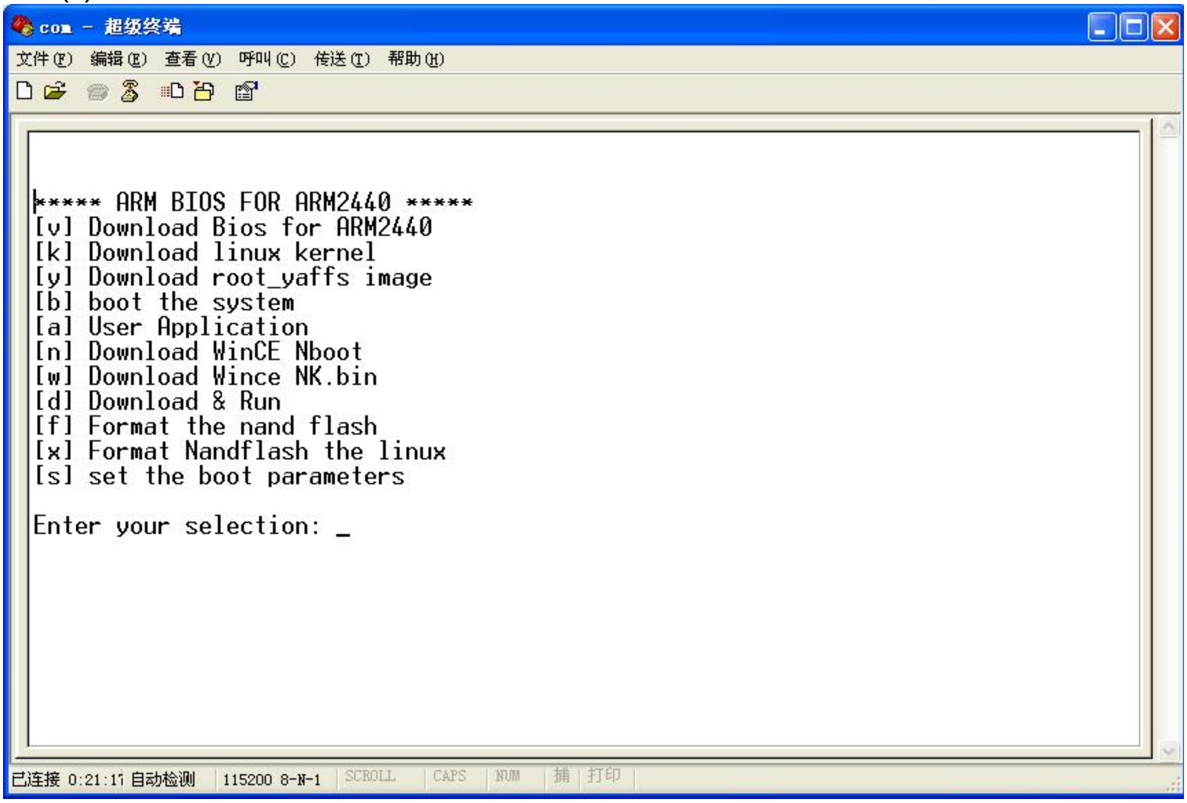
(4)安装文件系统

下面是详细的步骤。

3.1.1 格式化（分区）

提示：**分区将会擦除 Nand Flash 里面的所有数据**

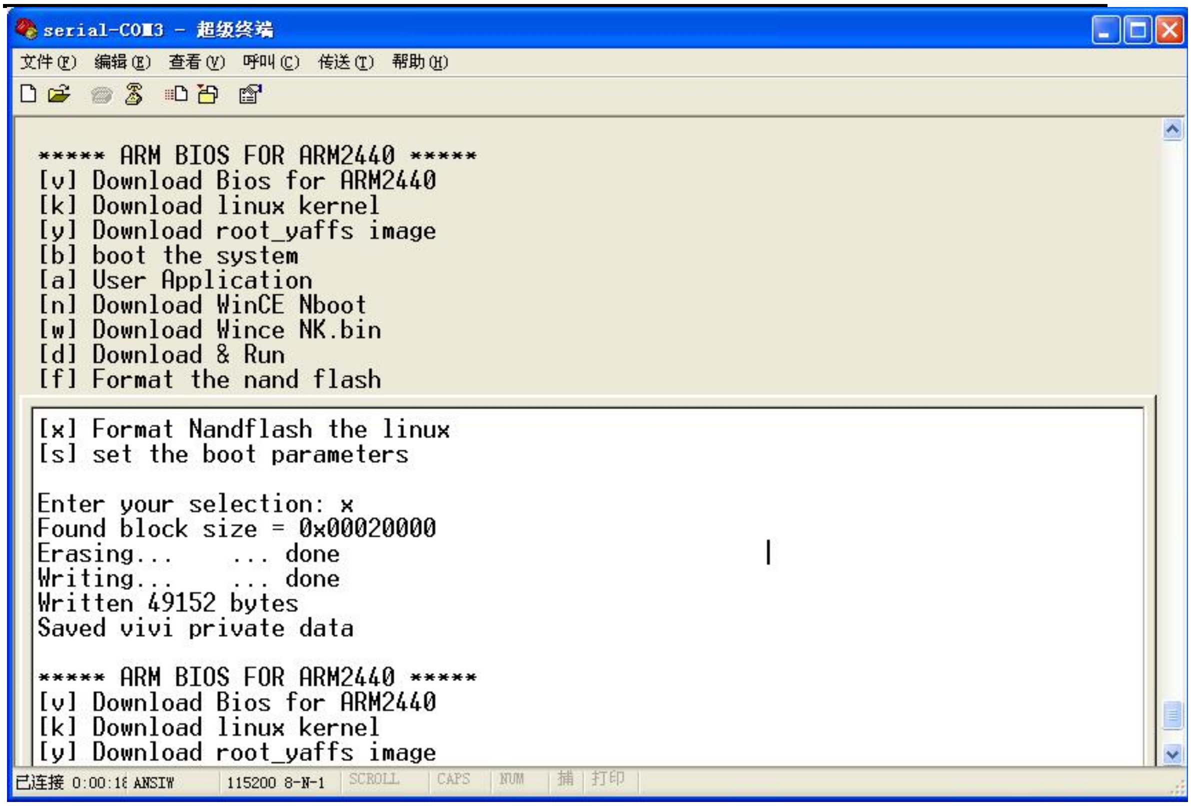
(1) 连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单：



(2) 选择功能号**[x]**开始对Nand Flash 进行分区，如图所示。

说明：有的 Nand Flash 分区时会出现坏区报告提示，因为armvivi 会对坏区做检测记录，因此这将不会影响板子的正常使用。

提示：普通的Nand Flash 并不能保证所有扇区都是完好的，如果有坏区，系统软件会对它们做检测处理，而不会影响整个软件系统的使用。保证完全无坏区的Nand Flash 另有型号，这那种Flash 一般没有现货，而且订货周期长，价格昂贵，一般场合很少用到；其他品牌的Nand Flash 也与此类似。



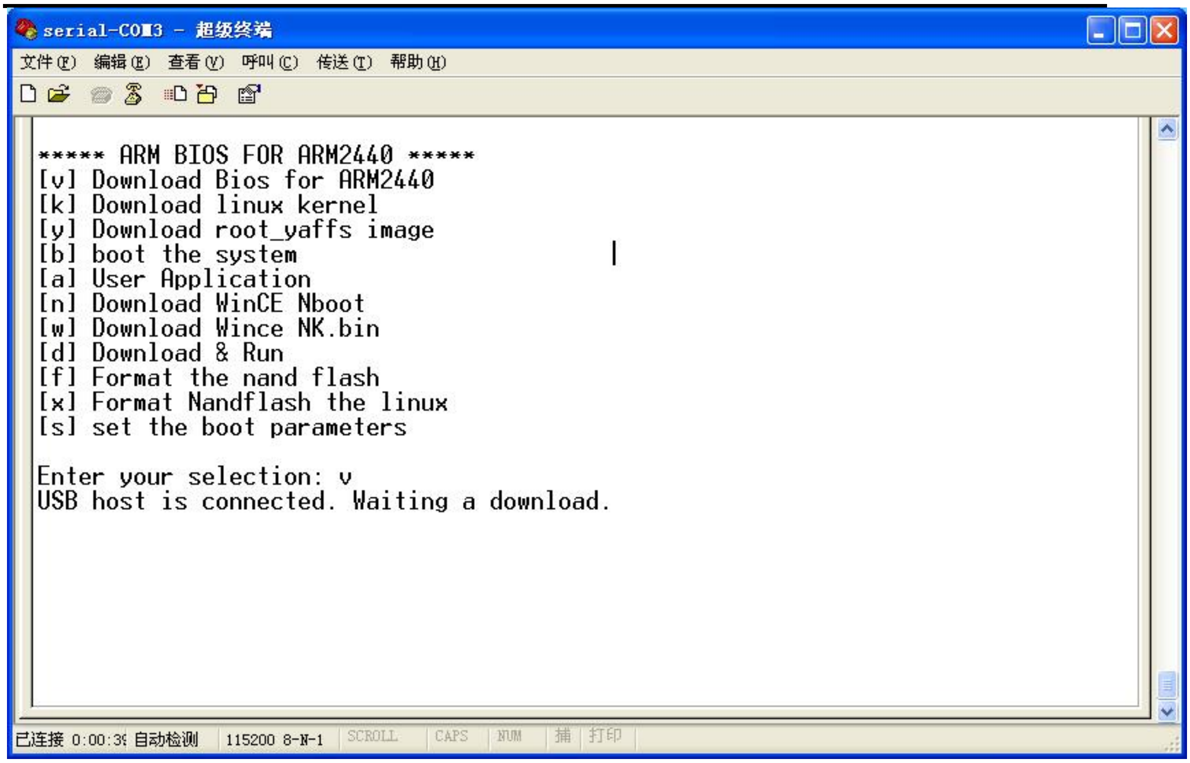
3.1.2 安装 bootloader

我们针对 Linux 系统提供了两种bootloader: vboot 和armvivi。

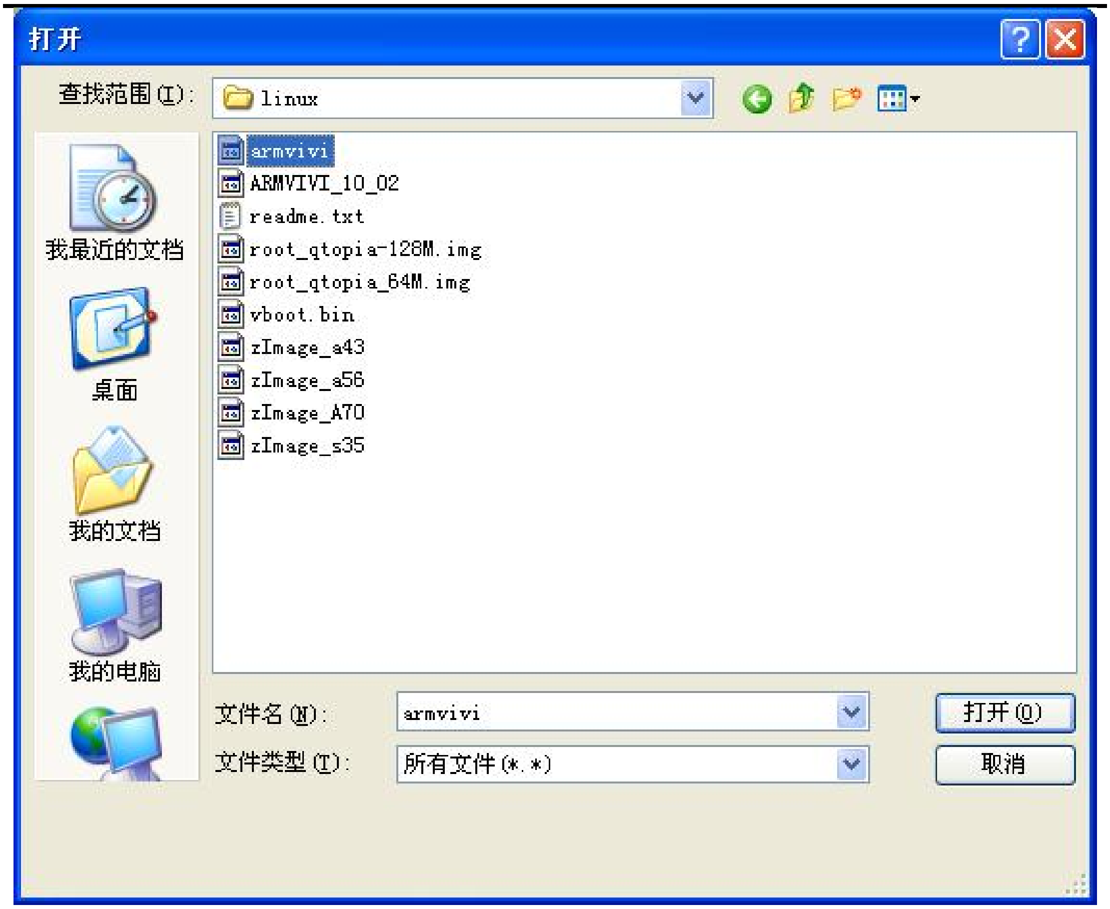
vboot 是一个十分简易的开源软件，它可以兼容启动64M/128M-1Gb Nand Flash 版GQ2440。armvivi是由三星原厂的vivi发展而来。

(1) 打开DNW 程序，接上USB 电缆，如果DNW 标题栏提示[USB: OK]，说明USB 连接成功，这时根据菜单选择功能号[v]开始下载armvivi。


DNW v0.60C - For WinCE [COM:x][USB:OK][ADDR:0x30000000]
Serial Port USB Port Configuration Help



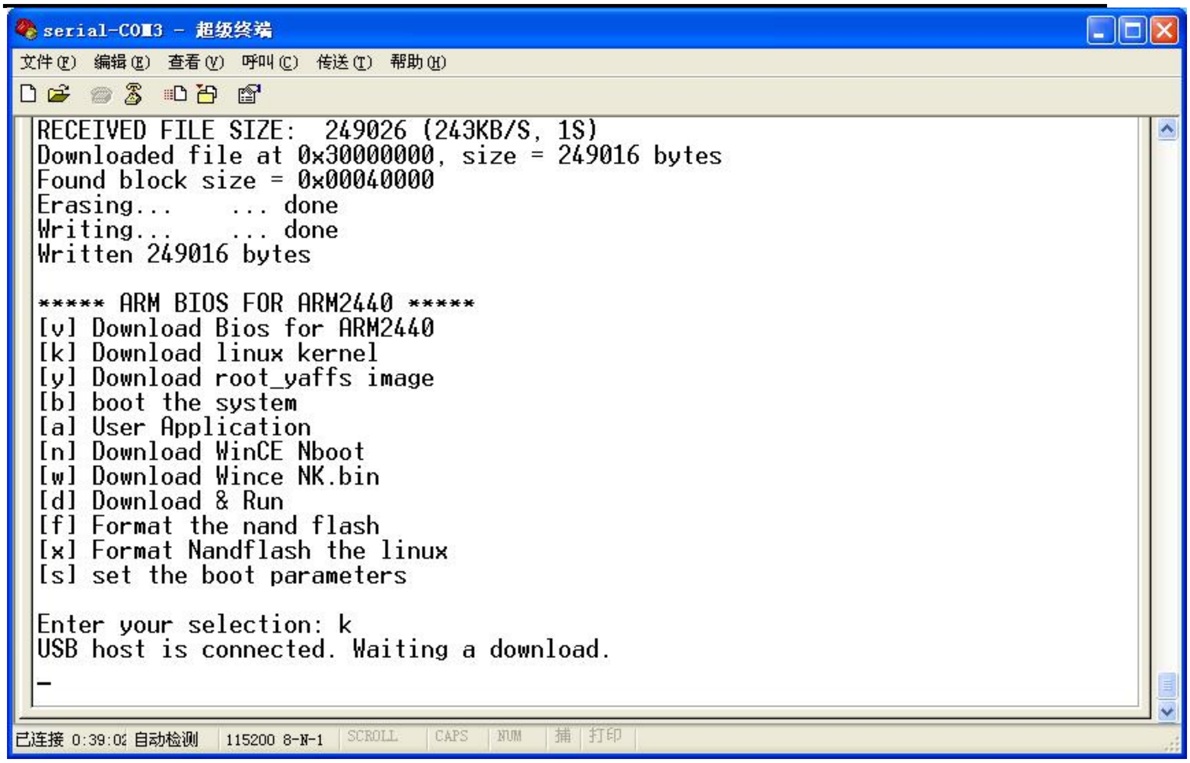
(2) 点击"USB Port->Transmit/Restore"选项，并选择打开文件armvivi(该文件位于光盘的images/linux/目录)开始下载。



(3) 下载完毕，BIOS 会自动烧写armvivi到Nand Flash 分区中，并返回到主菜单。

3.1.3 安装 Linux 内核

(1)在BIOS 主菜单中选择功能号[k]，开始下载linux 内核zImage。



(2) 点击"USB Port->Transmit"选项，并选择打开相应的内核文件zImage(该文件位于光盘的images\linux\目录)开始下载。

内核文件说明：

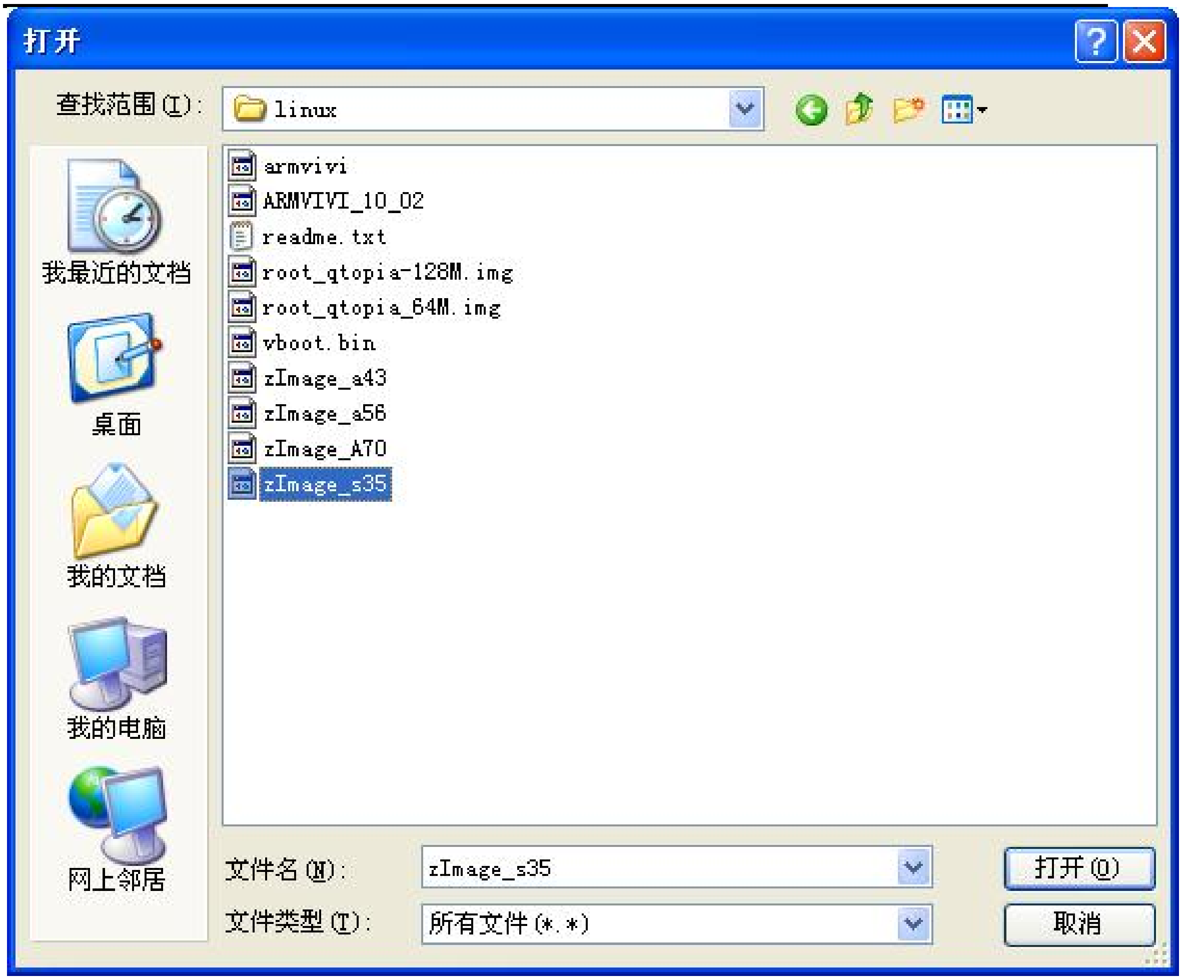
zImage_s35 – 适用于三星3.5"LCD

zImage_a43 – 适用于群创4.3"LCD

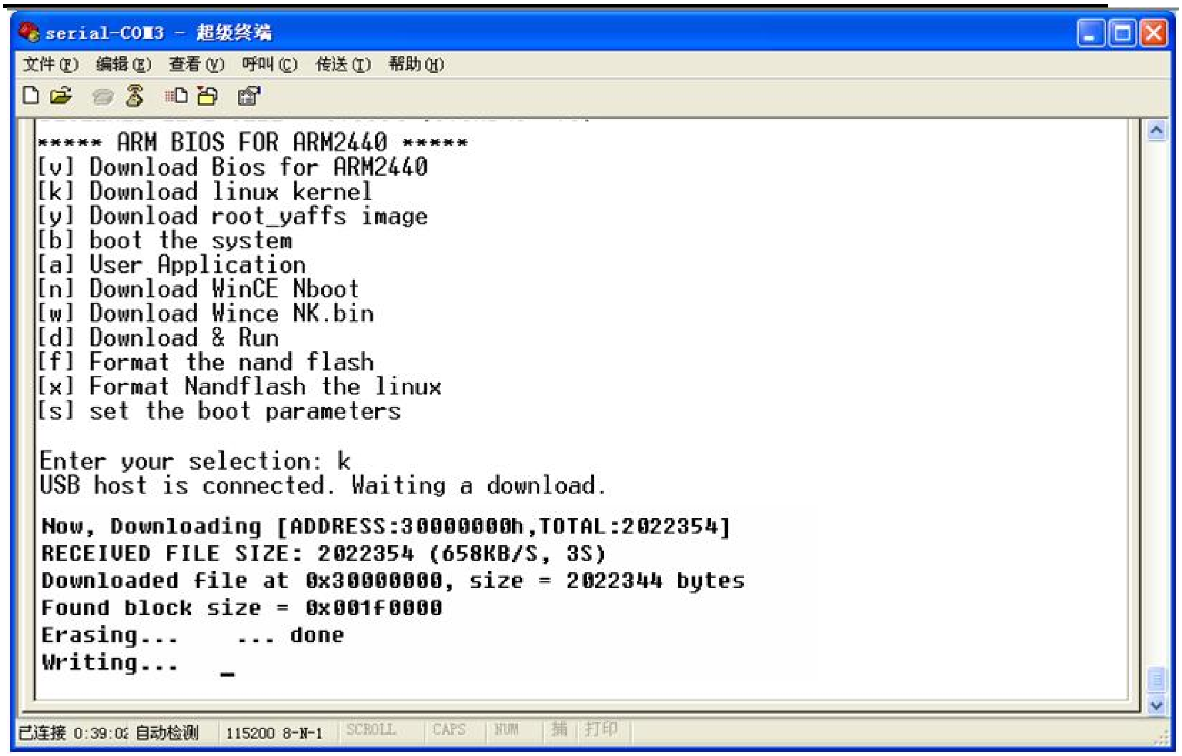
zImage_a56 – 适用于群创5.6"LCD(或兼容)

zImage_A70 – 适用于群创7 寸真彩屏，分辨率为800x480

实际可能与此不完全相同，请参考images\linux 目录下的readme.txt 文件说明。



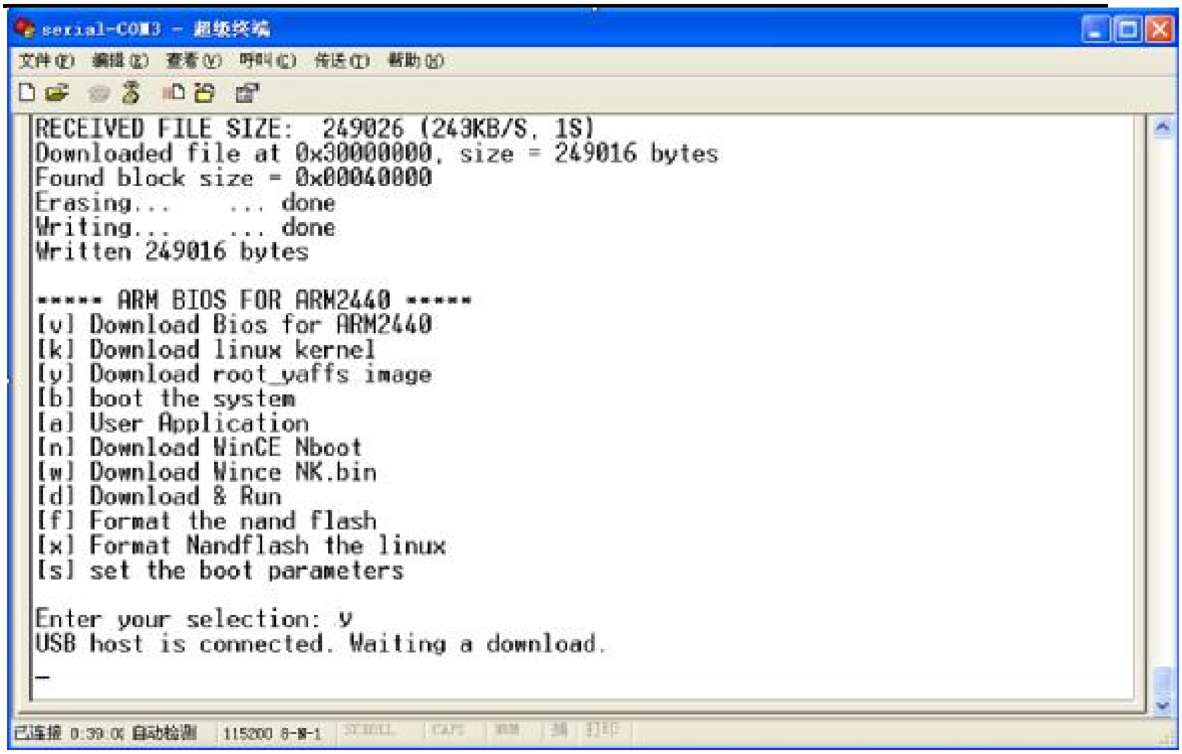
(3) 下载完毕，BIOS 会自动烧写内核到Nand Flash 分区中，并返回到主菜单，如图:



3.1.4 安装根文件系统

说明：针对 64M/128M-1Gb GQ2440，有不同的文件系统烧写映象文件：root_qtopia-64M.img 和 root_qtopia-128M.img，实际上它们的内容都是完全相同的，只是制作工具(mkyaffs2image)不同，我们把文件系统统称为 root-qtopia.img。

(1)在 BIOS 主菜单中选择功能号[y]，开始下载 yaffs 根文件系统映象文件



(2) 点击“ USB Port->Transmit/Restore ” 选项， 并选择打开相应的文件系统映象文件 root_qtopia.img(该文件位于光盘的images\linux 目录)开始下载。

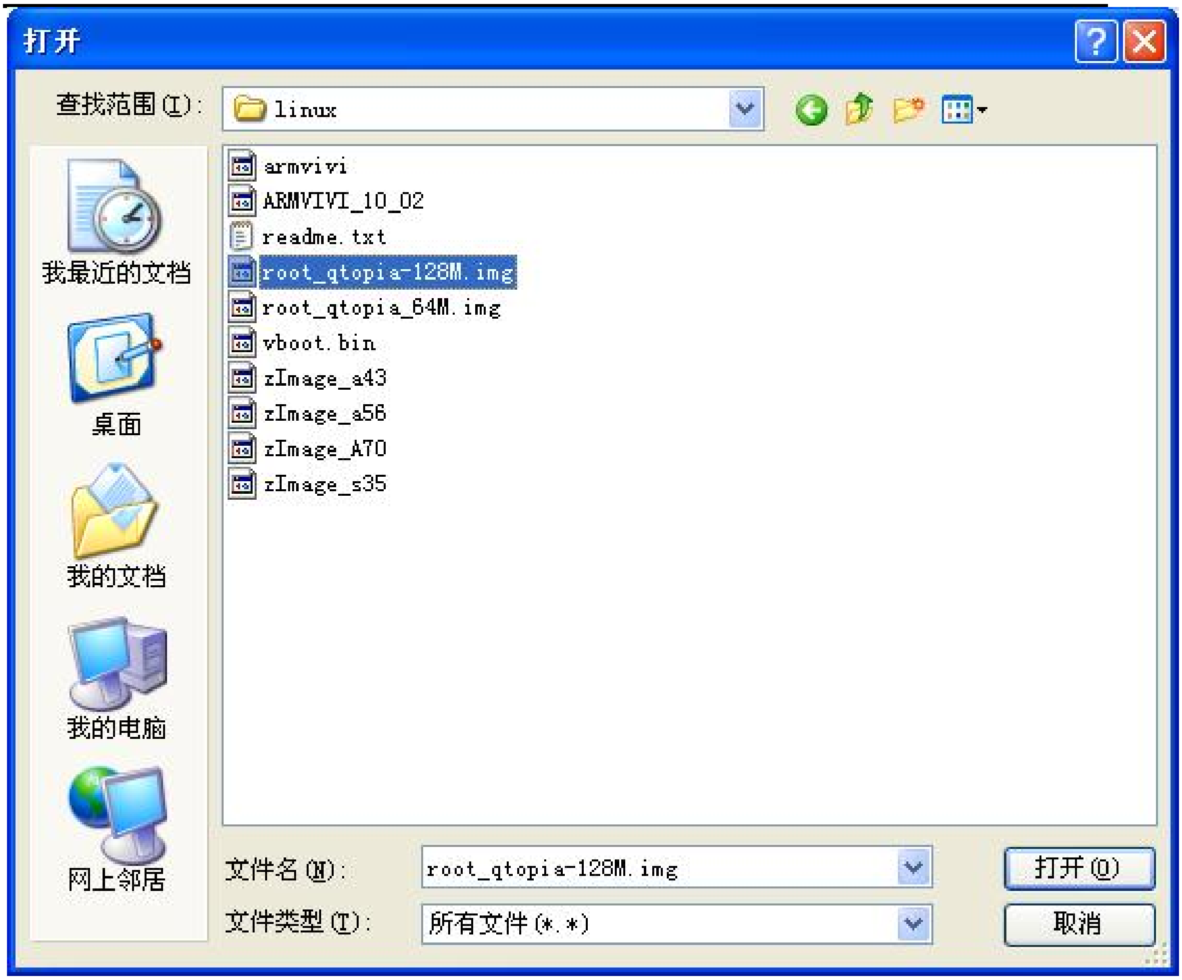
根文件系统映象文件说明:

root_qtopia_128M.img 支持128M以上的NandFlash。

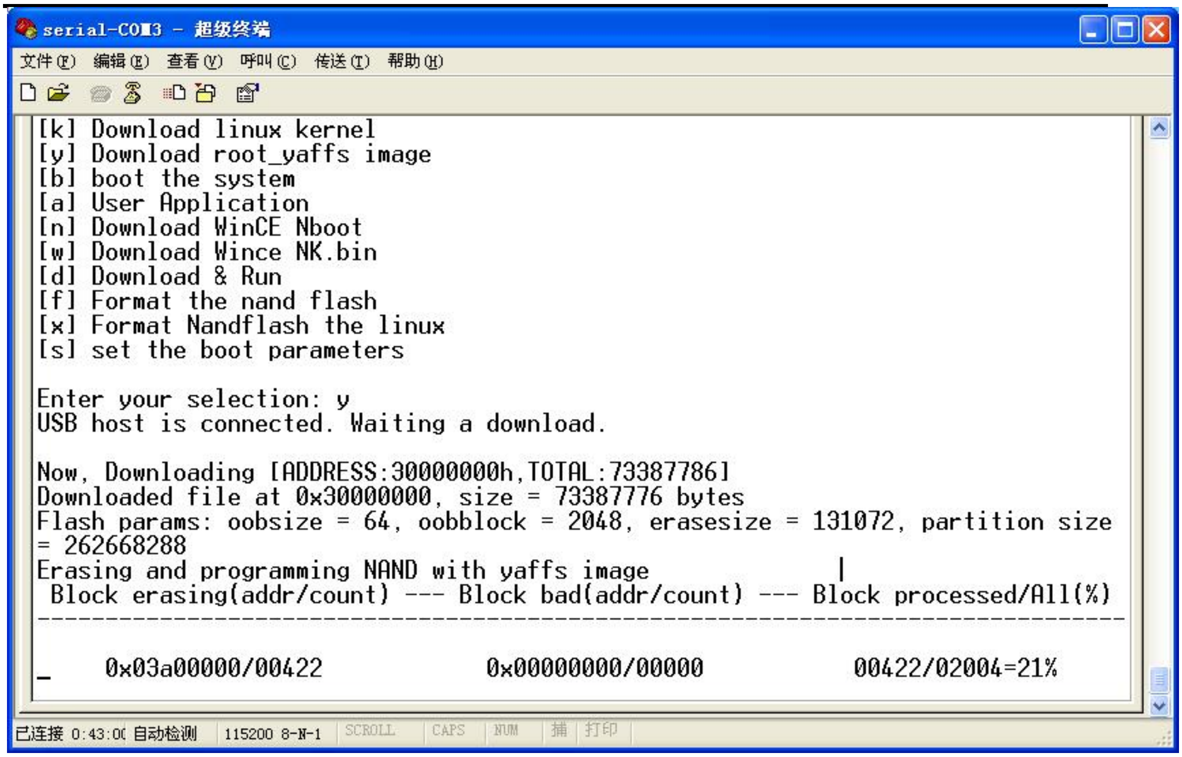
- 缺省安装的文件系统映象文件，可以同时支持USB鼠标和触摸屏，并自动识别VGA 模块输出和NFS启动

root_qtopia_64M.img 支持64M的NandFlash。

- 缺省安装的文件系统映象文件，可以同时支持USB鼠标和触摸屏，并自动识别VGA 模块输出和NFS启动。

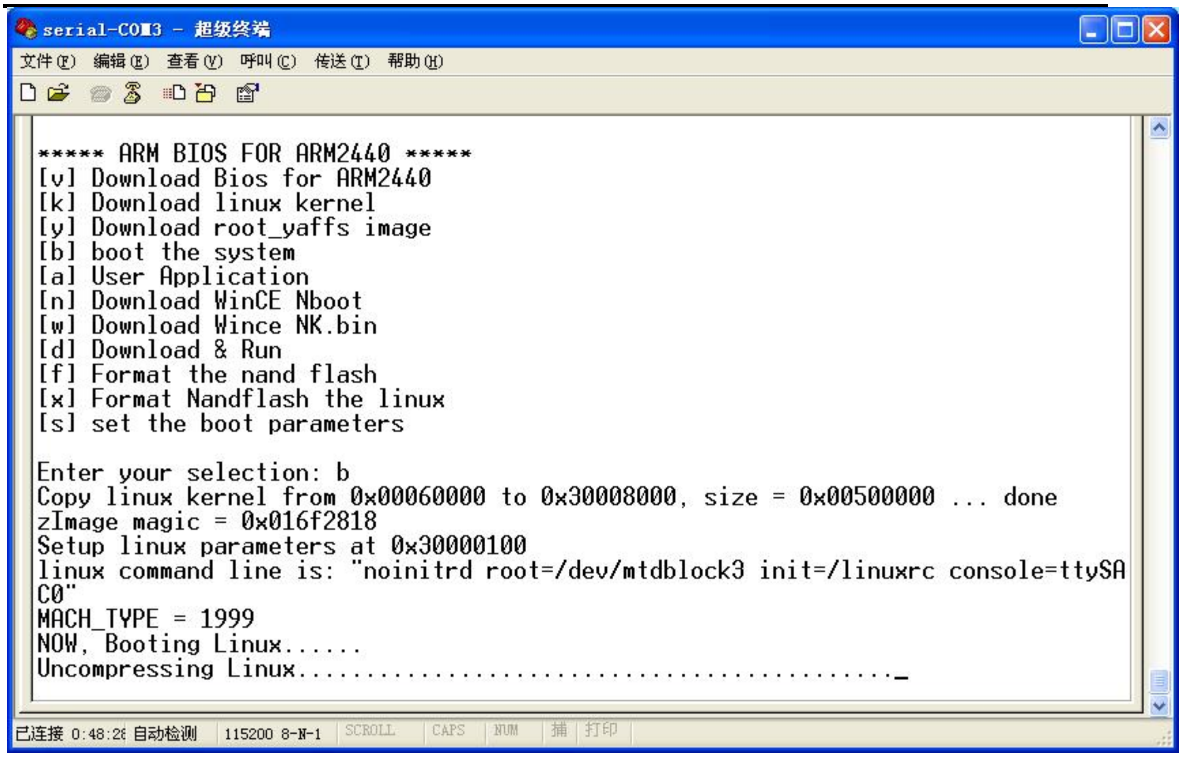


(3) 下载过程如图所示，下载完毕，BIOS 会自动烧写内核到Nand Flash 分区中，并返回到主菜单，如图：



注意：**下载完毕，请拔下USB 连接线，如果不取下来，有可能在复位或者启动系统的时候导致您的电脑死机。**

在 BIOS 主菜单中选择功能号[b]，将会启动系统。



如果您使开发板从Nand Flash 启动，则系统会在上电后自动启动。

3.2 安装 WinCE 系统

注意：**本小节假定您已经按照前面的方法安装了 USB 驱动，并把开发板设置为 NOR Flash 启动，系统更新和安装完毕请设置为 Nand Flash 启动，设置方法请参考前面的章节。**

说明:
安装 WinCE5 所需要的二进制文件位于光盘的"\images\wince5.0"目录中
安装 WindowsCE 系统主要有以下步骤:
(1)安装 bootloader
(2)安装 BootLogo(bmp 格式)
(3)安装 WindowsCE 内核映象
下面是详细的步骤:
提示: **请先连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单**

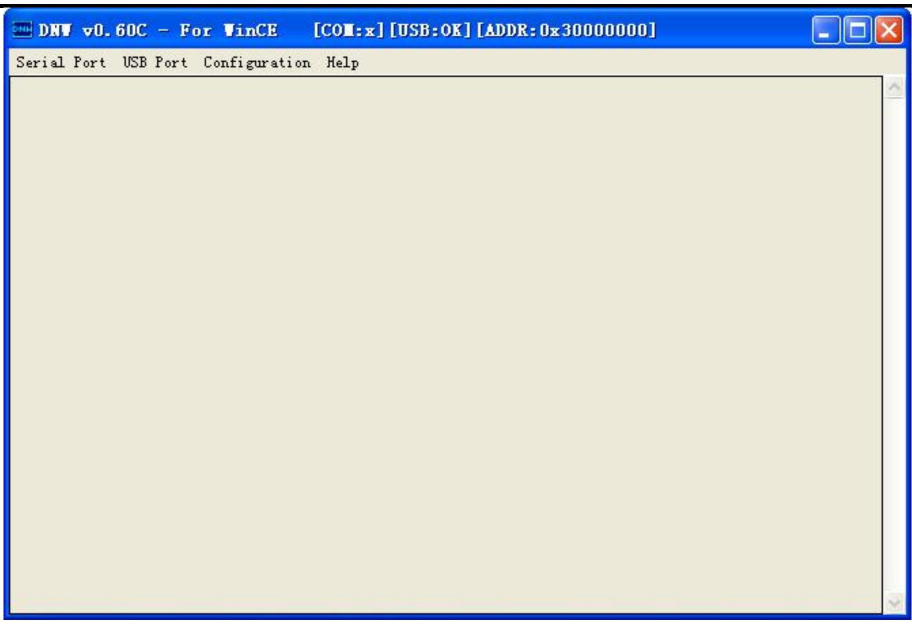
3.2.1 安装 Bootloader

本开发板提供启动wince的bootloader 是 nboot.bin，其源码位于**\GQ2440\WinCE源代码和软件\WindowsCE5.0\nboot**目录下，它具有一下特点:

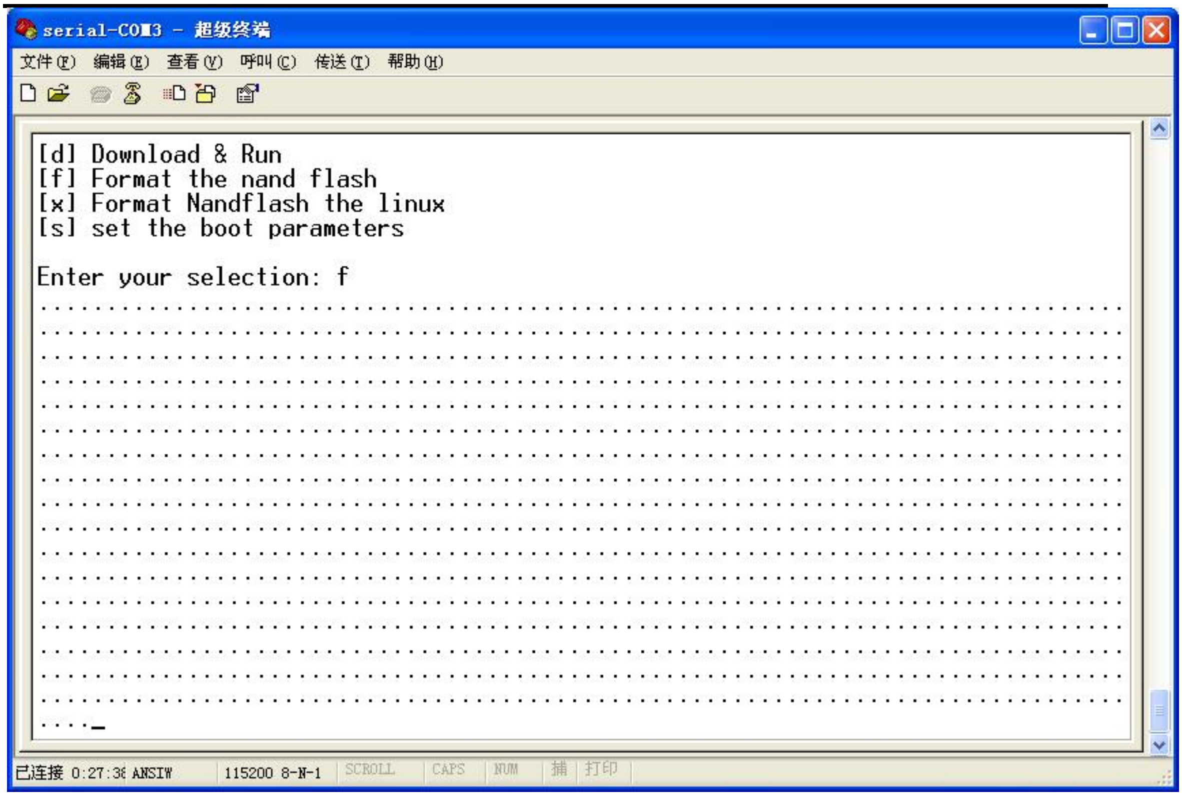
- 支持快速启动WindowsCE(一般6-10 秒，视内核大小而定)
- 支持开机Logo(通过USB下载普通的bmp格式真彩图片)
- 带进度条(用户可以根据源代码配置文件自定义进度条的各个属性: 颜色、长宽、起始位置等)

下面以 nboot_s35.bin 为例(以下简称Nboot.bin)，介绍Nboot.bin 的下载烧写步骤:

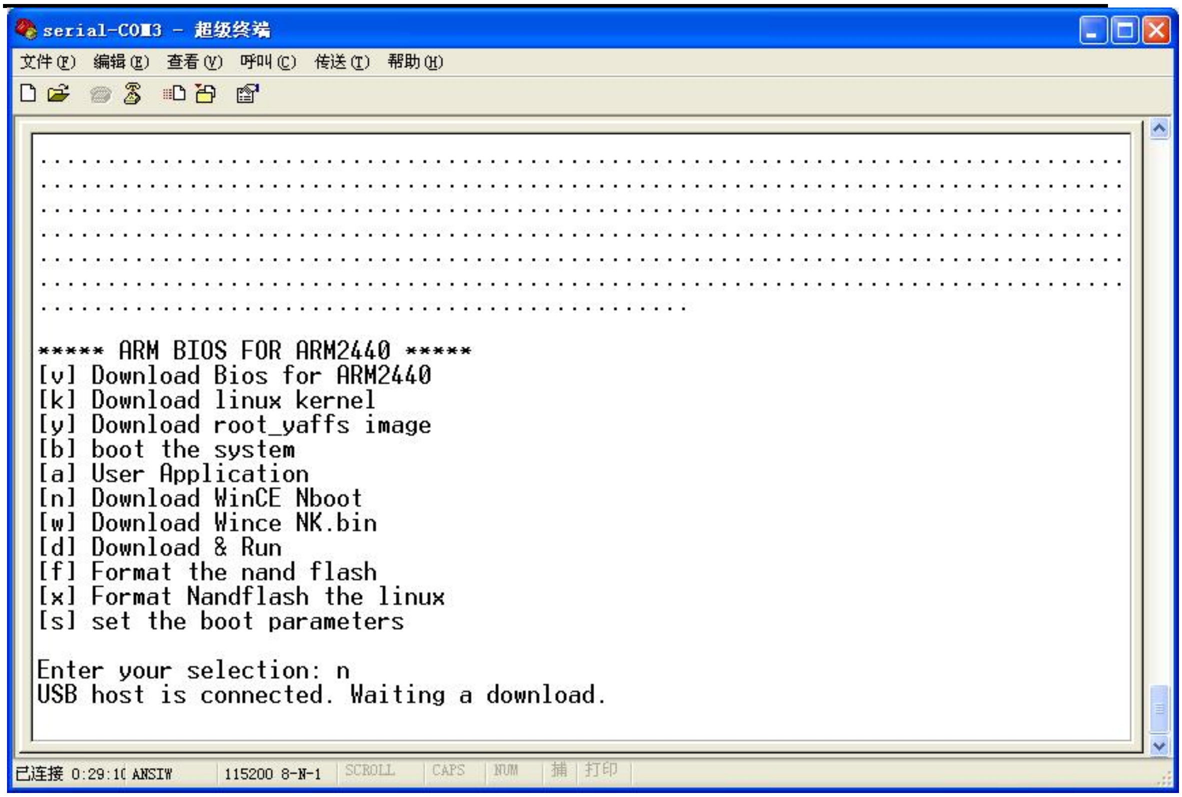
(1) 打开DNW 程序，接上USB 电缆，如果DNW 标题栏提示[USB: OK]，说明USB 连接成功，在下载nboot之前，应该先格式化分区。


DNW v0.60C - For WinCE [COM:x] [USB:OK] [ADDR:0x30000000]
Serial Port USB Port Configuration Help

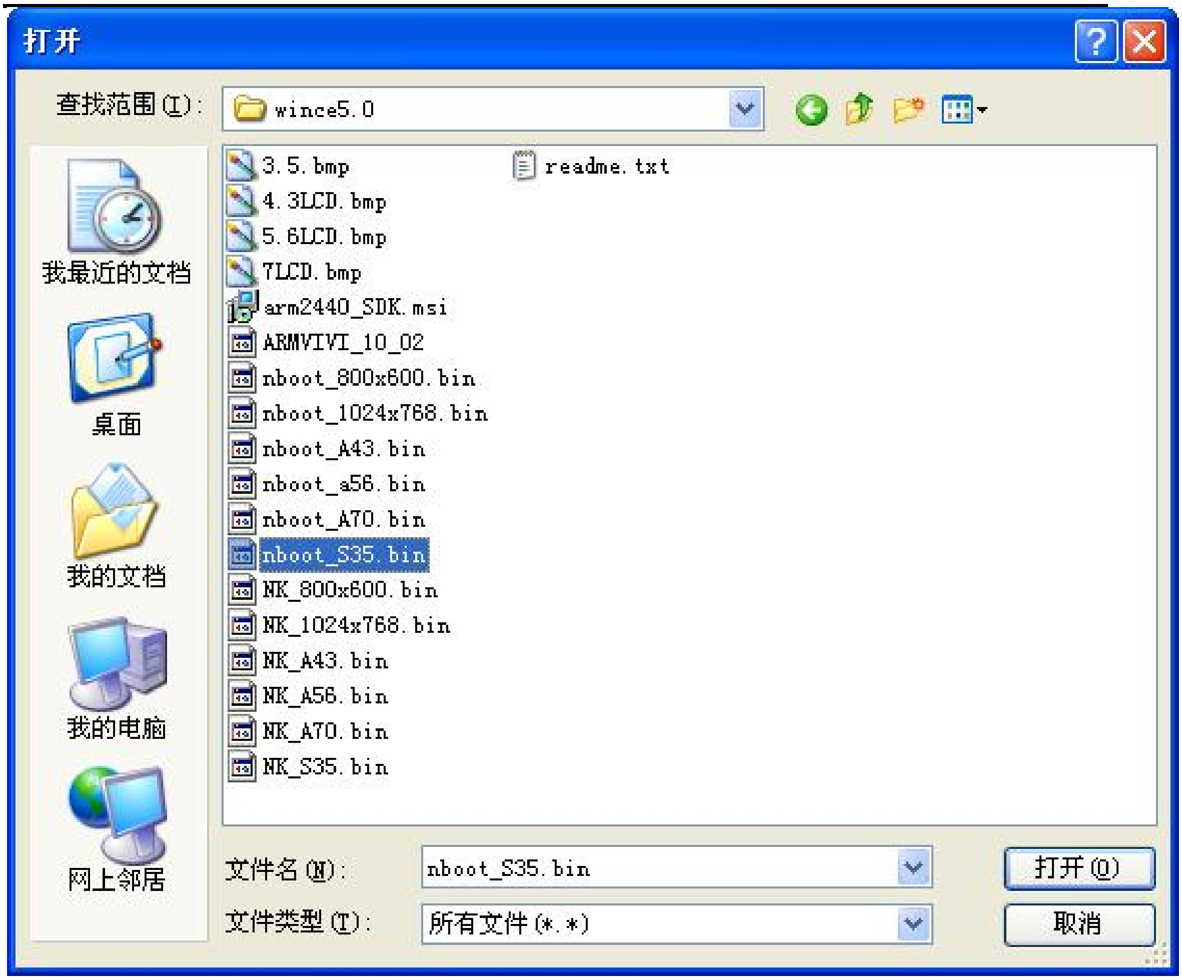
敲入字符f，格式化分区：



敲入菜单选择功能号[n]开始下载Nboot.bin:



(2).点击“USB Port->Transmit”选项，并选择打开文件Nboot.bin(该文件位于光盘的\images\wince5.0 目录)开始下载。



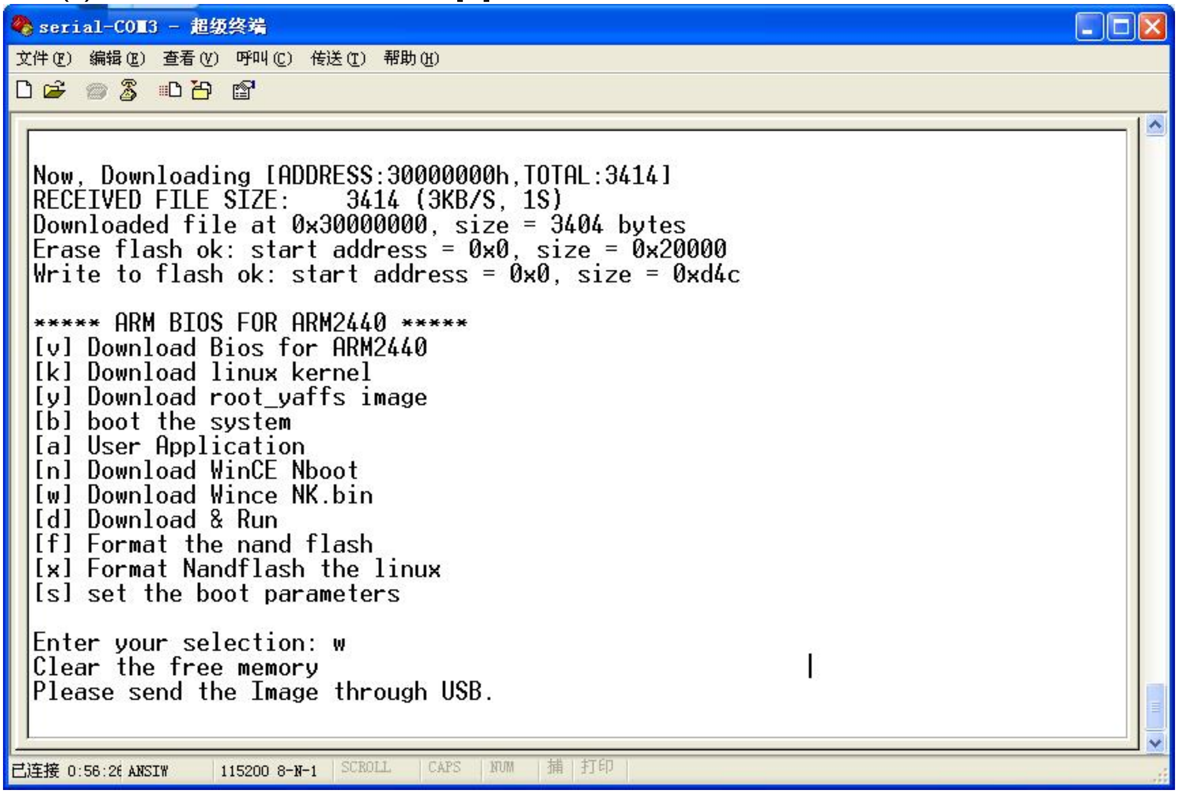
(4)下载完毕，BIOS 会自动把Nboot.bin 烧写到Nand Flash 的Block 0。

3.2.2 安装 wince 内核映象

说明：armvivi 已经集成了WinCE 内核烧写功能，不需要加载其他的跳板文件(诸如Eboot等)，armvivi 先通过USB 下载NK.bin 到内存中，然后格式化Nand Flash，再创建

系统分区(NK.bin 所占用的NandFlash 空间，BINFS 格式，为只读)和用户分区(对应于WinCE启动后的ResidentFlash 目录，FAT 格式，可读写)

(1)在BIOS 主菜单中选择功能号[w]，开始下载WINCE 内核:



(2) 点击"USB Port->Transmit/Restore"选项，并选择打开相应的内核文件NK.bin(该文件位于光盘的"\images\wince5.0"目录或者"\images\wince5.0"目录)开始下载。

针对不同型号LCD，nboot镜像文件说明如下:

nboot_a70.bin - A70 表示支持群创7"LCD显示屏

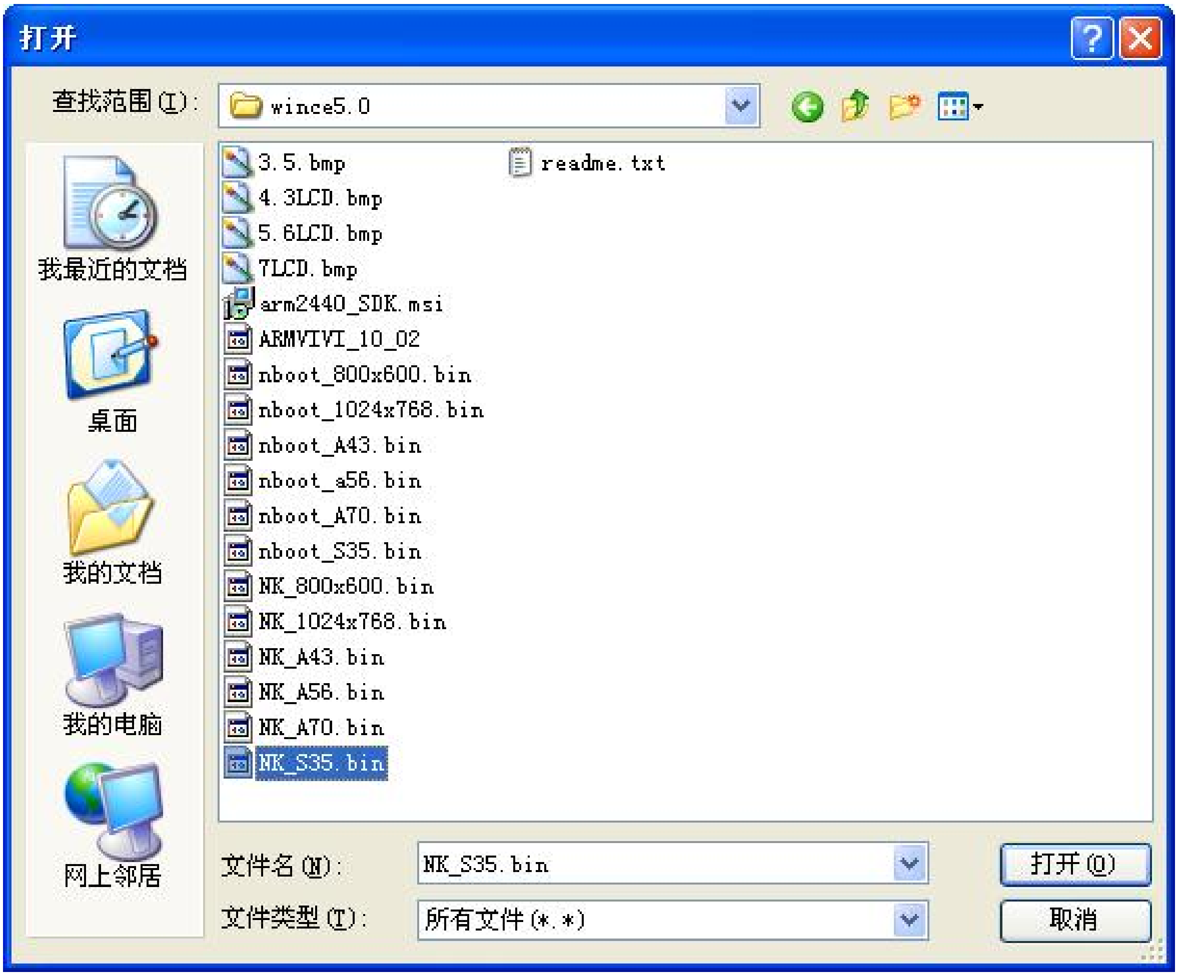
nboot_a43.bin - A43 表示支持群创4.3"LCD显示屏

nboot_s35.bin - s35 表示支持三星3.5"LCD显示屏

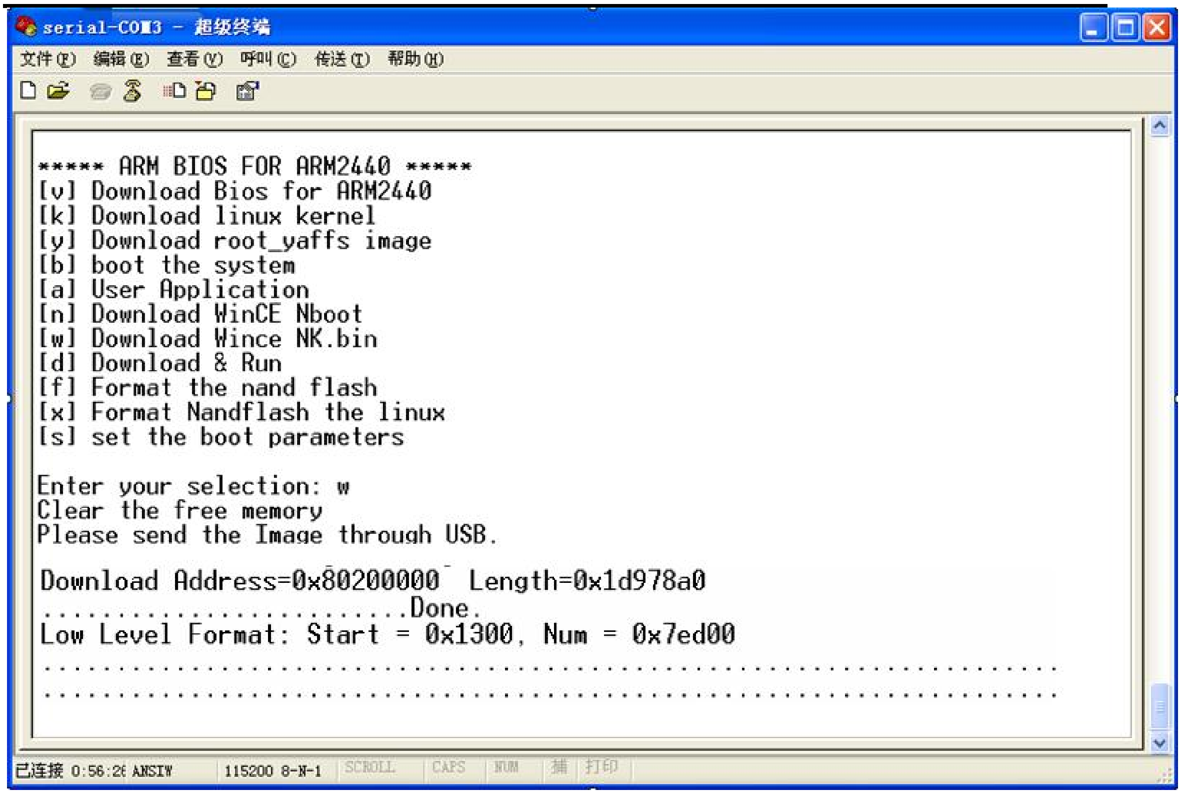
nboot_a56.bin - A56 表示支持群创5.6"LCD显示屏

实际可能与此不完全相同，请参考"images\wince5.0"或者"images\wince6.0"目录下的

readme.txt 文件说明。



下载完毕，BIOS 会开始格式化Nand Flash，并依次创建分区烧写WinCE 内核文件，烧写完毕，会自动启动WinCE 系统，整个过程串口信息如下图：



3.3 下载到内存运行

armvivi 支持很多程序下载到内存中运行，例如 2440test、uCos2、Linux 内核、WinCE 内核等。对于简单的系统，一般可以通过 USB 下载到内存中运行调试，对于比较高级完善的系统则烧写到 Nand Flash 中运用其本身的功能进行进一步的调试。

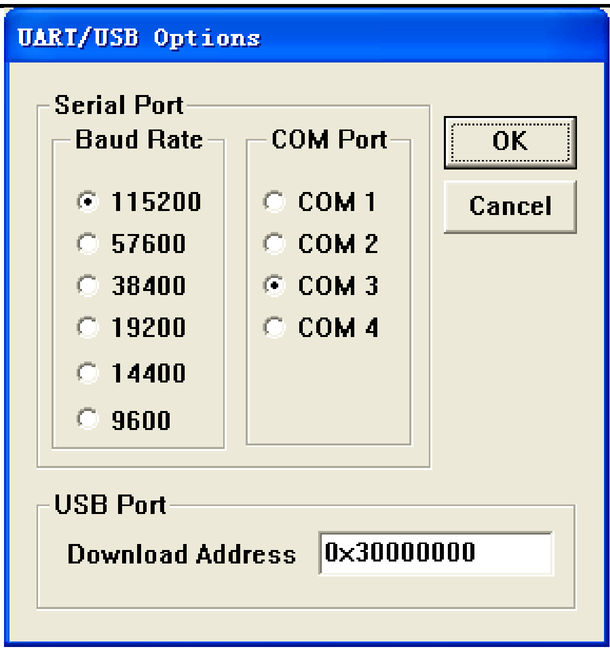
3.3.1 运行 2440test

(1)连接好开发板电源，串口线，USB线，并通过Nor_KEY按钮使开发板从Nor Flash启动，分别打开串口超级终端和DNW，上电启动开发板。

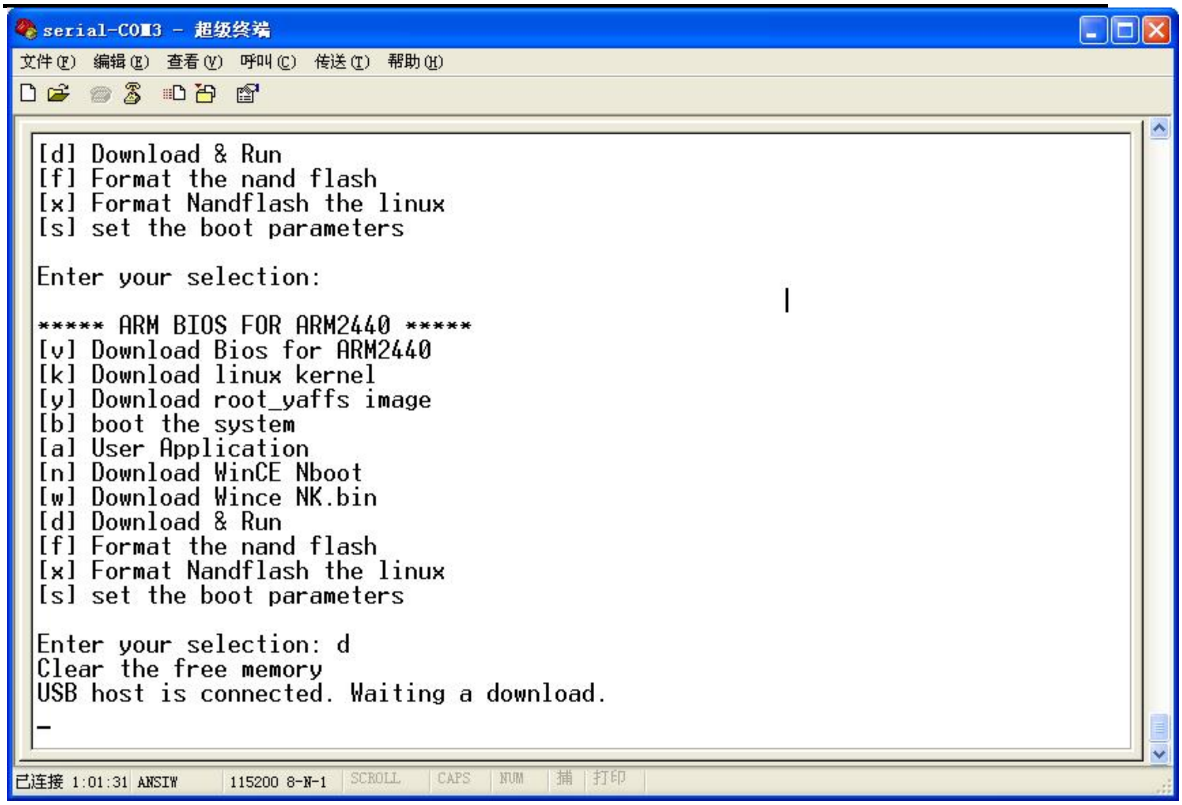
(2)保证USB 驱动已经安装好(前面已经详细介绍了USB 驱动的安装方法)，这时可以看到DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：



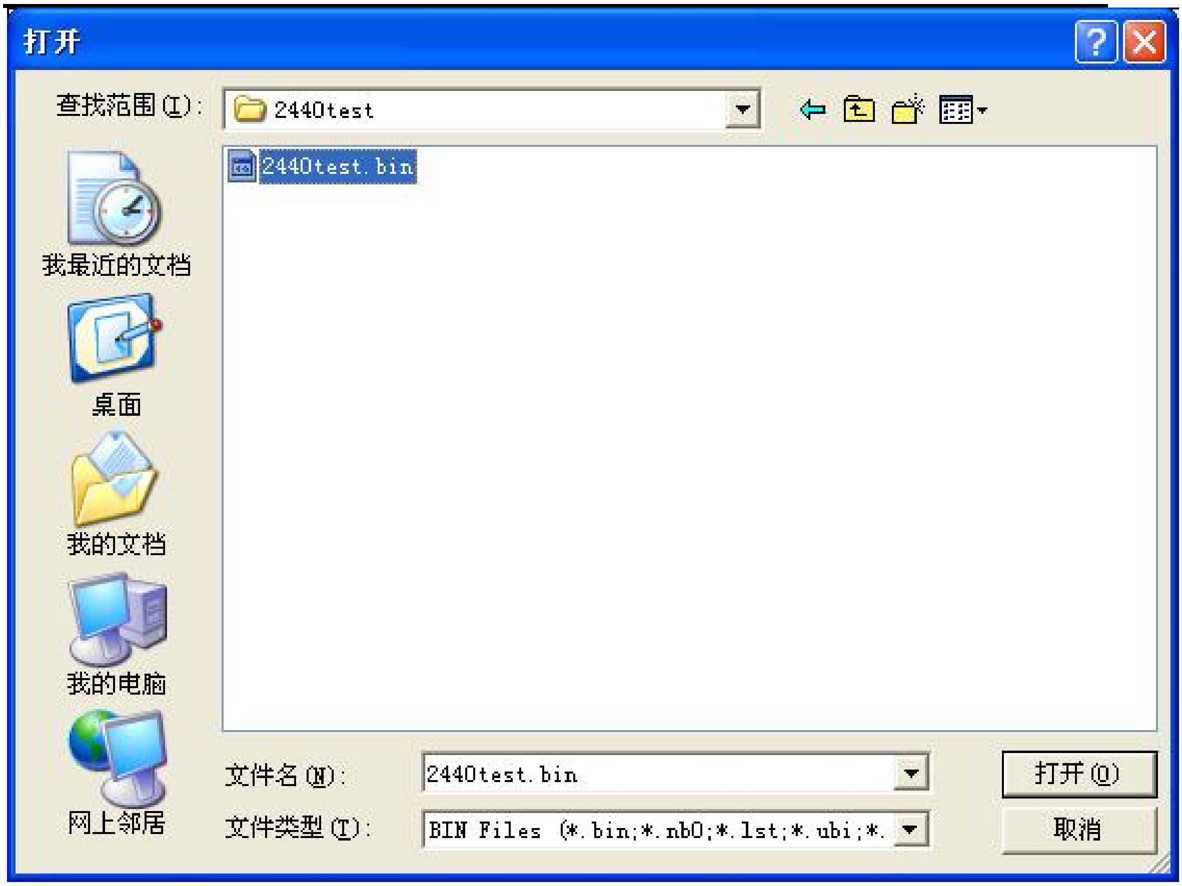
(3).点DNW 菜单Configuration，设置USB 下载运行地址为0x30000000



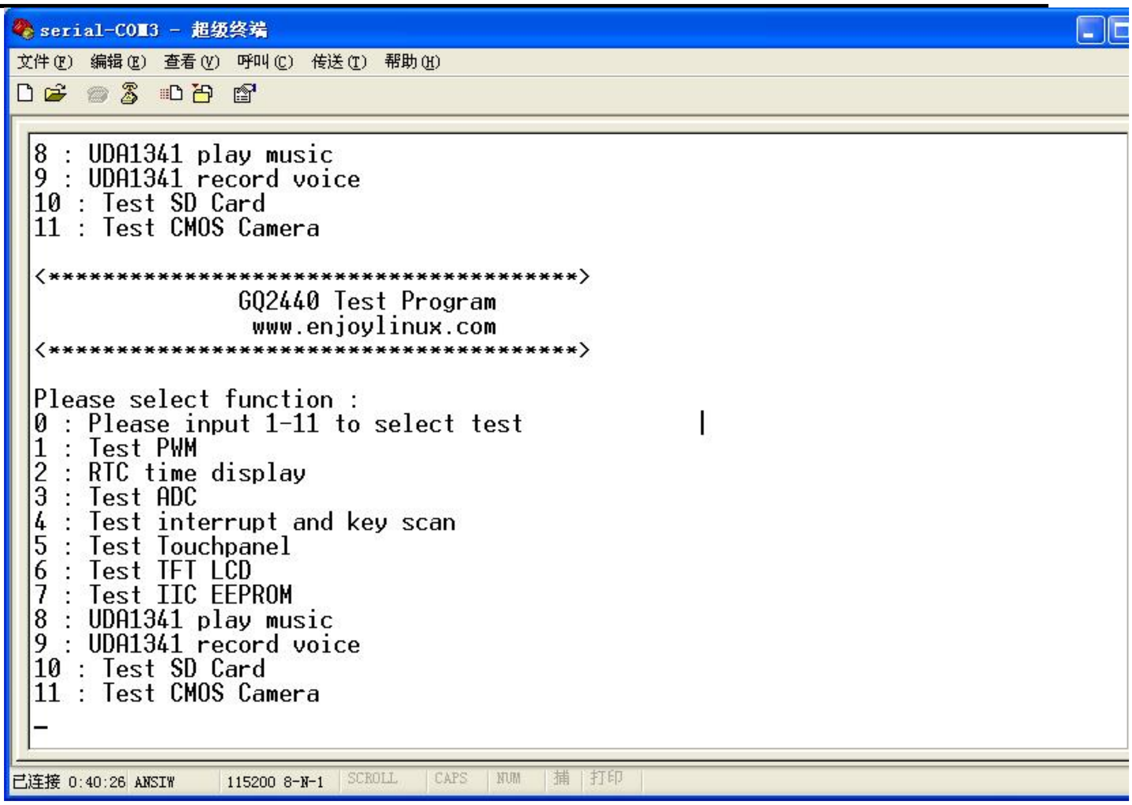
(4)这时在超级终端的BIOS 功能菜单中选择功能号[d]，出现USB下载等待提示信息:



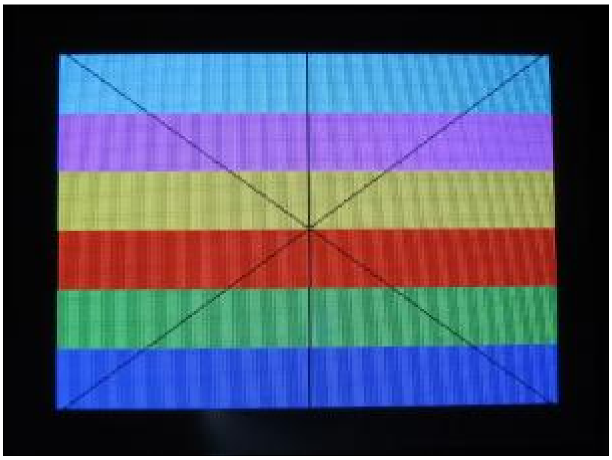
(5)点击 DNW 程序的【USB Port】|【Transmit】，选择 2440test.bin 这个映象文件(在光盘的 images\2440test\目录下面)，接着点"打开"，这样就开始下载了。



(6)下载结束后，会自动运行，终端出现如下界面。


serial-COM3 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
8 : UDA1341 play music
9 : UDA1341 record voice
10 : Test SD Card
11 : Test CMOS Camera
<***>
GQ2440 Test Program
www.enjoylinux.com
<***>
Please select function :
0 : Please input 1-11 to select test
1 : Test PWM
2 : RTC time display
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test TFT LCD
7 : Test IIC EEPROM
8 : UDA1341 play music
9 : UDA1341 record voice
10 : Test SD Card
11 : Test CMOS Camera
已连接 0:40:26 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

LCD 出现彩带图形



当程序下载完成后，开发板会自动执行该测试程序。裸机测试程序也是一键式菜单。

说明：2440test 运行起来之后，armvivi 就失去了对 CPU 的控制权，而 2440test 本身是没有 USB 部分支持的，因此此时会出现 USB 断开的情况。

3.3.2 运行 Linux

说明：在内存中运行 linux 系统，一般是指linux 内核(具体为zImage 文件)，文件系统是无法通过网络或者USB 下载到内存中运行的。一般是借助于linux 的启动命令，指定NFS (网络文件系统)，或者使用开发板“本地”文件系统，如yaffs(可通过armvivi 的“y”命令烧写root_default.img 或者其他文件系统映象文件)。

如何通过 linux 命令指定NFS 启动系统？

在本开发板中，首先进入armvivi 菜单，按“q”键进入armvivi 的命令行模式，输入:

armvivi> param set linux_cmd_line "console=ttySAC0 root=/dev/nfs

nfsroot=192.168.1.5:/opt/Enjoylinux/GQ2440/root_qtopia

ip=192.168.1.230:192.168.1.5:192.168.1.105:255.255.255.0:www.enjoylinux.cn:eth0:on"

然后记得输入:

armvivi> param save

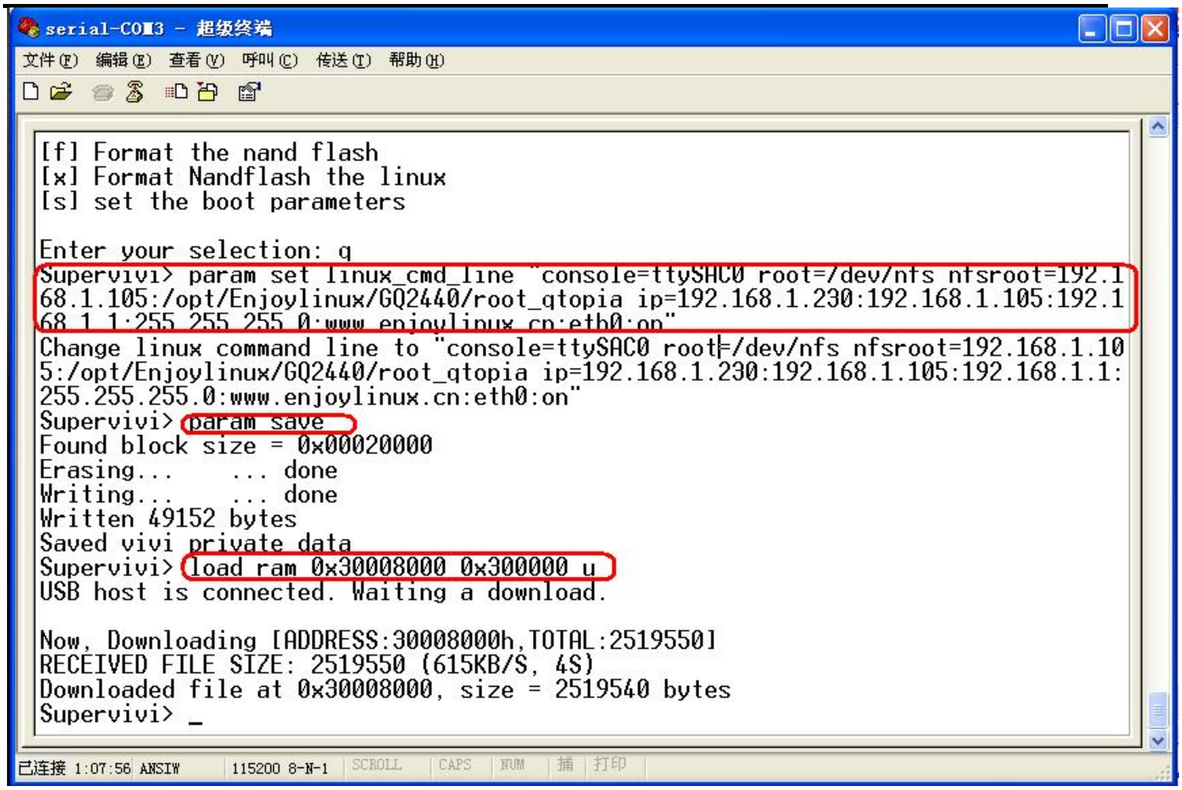
命令进行保存。(nfs服务还需要在pc端进行设置，请参见5.5节内容）

下面是使用 USB 下载linux 内核到开发板中，并启动运行的步骤，这里使用的是“本地”文件系统root_qtopia.img:

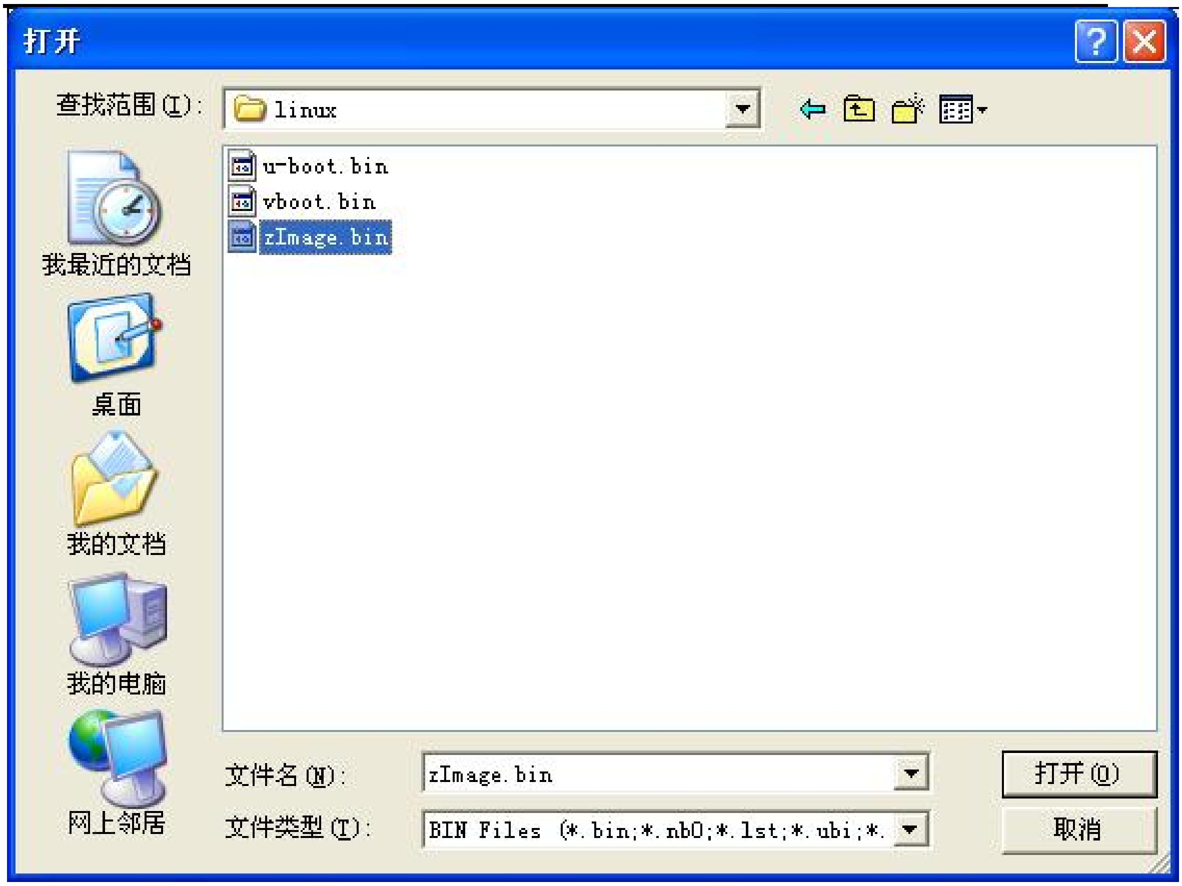
(1)连接好开发板电源，串口线，USB 线，并通过Nor_KEY按钮使开发板从Nor Flash 启动，上电启动开发板。

(2)保证USB 驱动已经安装好(前面已经详细介绍了USB 驱动的安装方法)，这时可以看到DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]。

(3)按[q]进入armvivi 的命令行模式，输入“load ram 0x30008000 0x300000 u”。



(4).点击DNW程序的【USB Port】|【Transmit】，选择zImage_t35 或者zImage_A70 这个映象文件(在光盘的images\linux 目录下面)，接着点“打开”，这样就开始下载了。



(5)下载完毕后，输入“boot ram”，启动内核。

第4章 ADS1.2 集成开发环境的使用

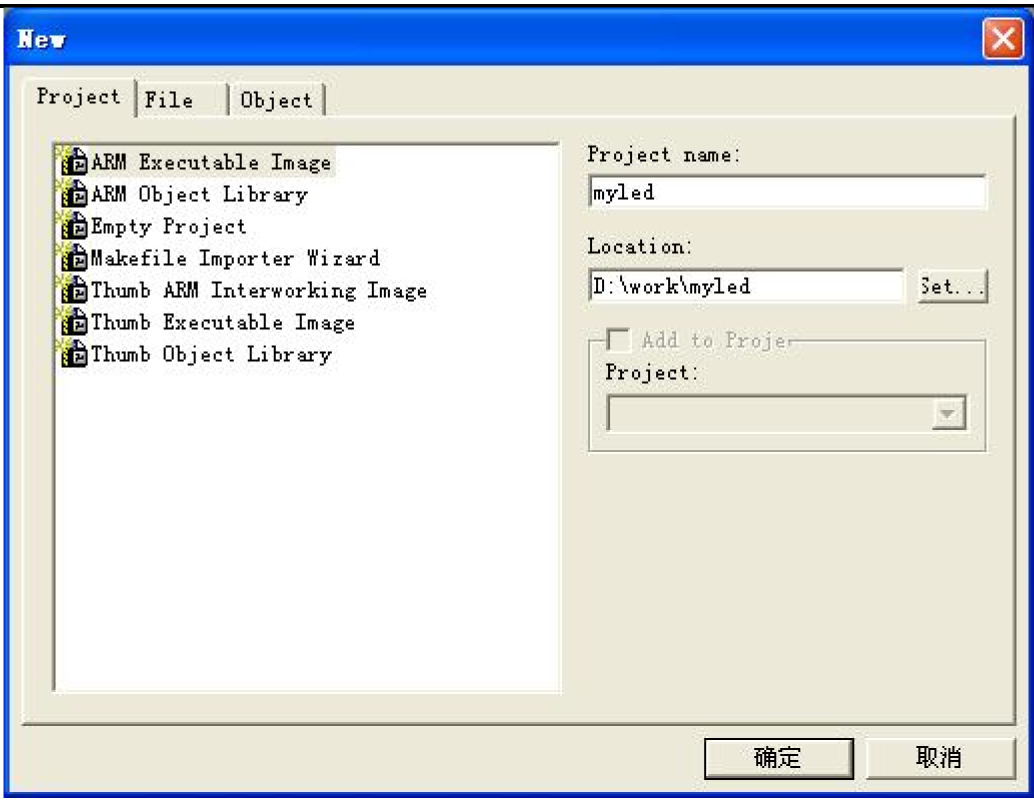
ARM ADS 的全称为 ARM Developer Suite，它是 ARM 公司推出的新一代 ARM 集成开发环境，我们使用的 ADS 为 1.2 版本，它取代了早期的 ADS1.1 和 ADS1.0，它可以安装在 WindowsNT/2000/98/95/XP 上面使用。

4.1 使用 ADS 创建 LED 工程

本节通过一个简单的具体实例，介绍如何使用 ADS 集成开发环境。包括如何创建一个新的工程，如何配置编译选项，并编译生成可以直接烧写到 Flash 中的 bin 格式二进制可执行文件。

4.1.1 建立一个工程

在 ADS 集成开发环境中，点File->New，打开如图所示窗口：



可以看到有 7 种工程类型可以选择:

ARM Excuteable Image: 用于由ARM 指令的代码生成一个ELF 格式的可以执行映象文件。

ARM Object Library: 用于由ARM 指令的代码生成一个armar 格式的目标文件库。

Empty Project: 用于创建一个不包含任何库或者源文件的工程。

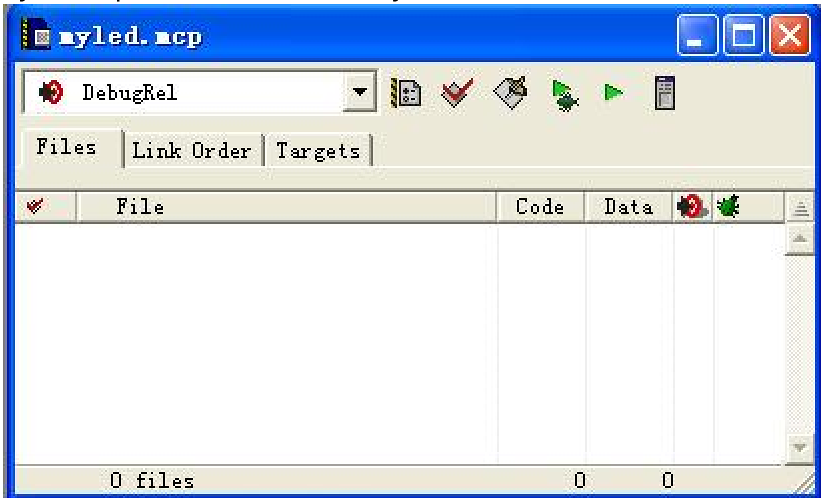
Makefile Importer Wizard: 用于将Visual C 的nmake 或者GNU make 文件转入到CodeWarrior IDE 工程文件。

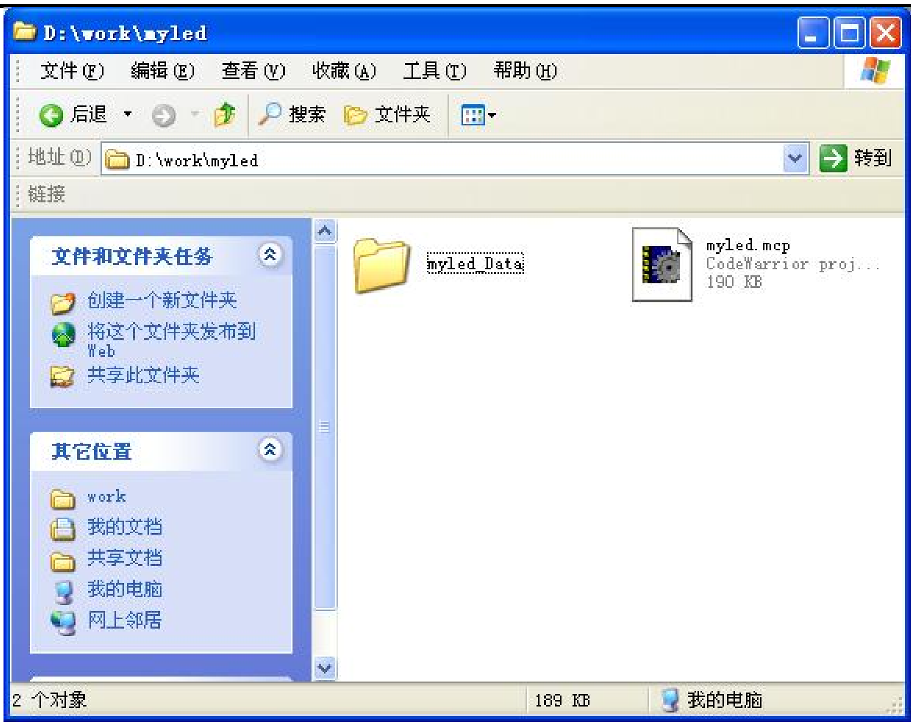
Thumb ARM Excutable Image: 用于由ARM 指令和Thumb 指令的混和代码生成一个可执行的ELF 格式的映象文件。

Thumb Excutable image: 用于由Thumb 指令创建一个可执行的ELF 格式的映象文件。

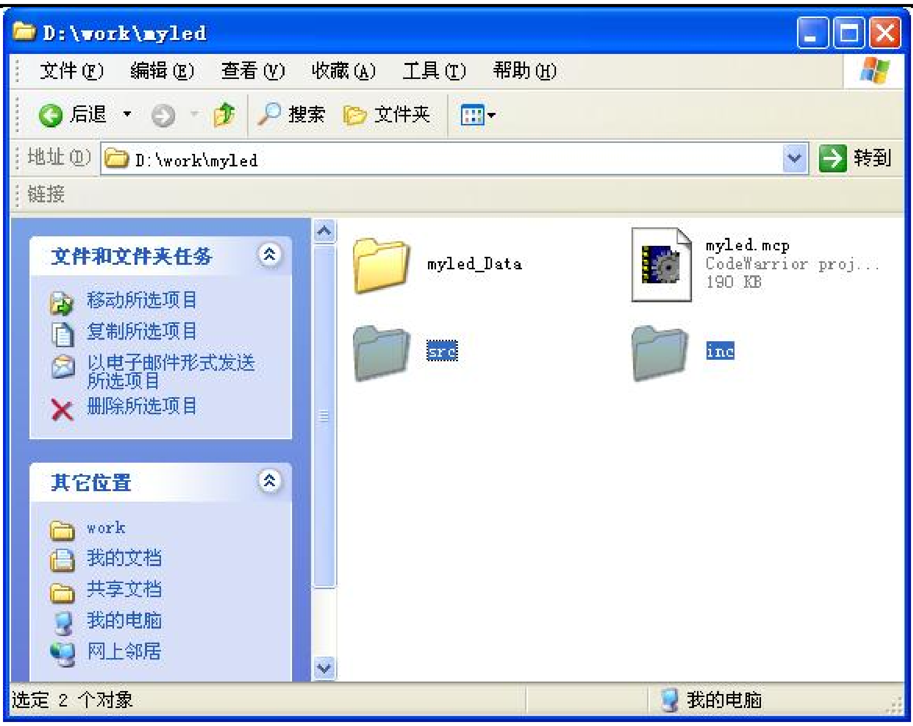
Thumb Object Library: 用于由Thumb 指令的代码生成一个armar 格式的目标文件库。

我们在这里选择 ARM Executable Image，在“Project name: ”中输入工程文件名，本例为“myled”，点击“Location: ”文本框的“Set”按钮，浏览选择想要保存该工程的路径(本例为“D:\work”)，将这些设置好之后，点击“确定”，即可创建一个新的名为myled 的工程。这个时候会出现 myled.mcp 窗口，如图所示，同时会在D:\work 目录下创建一个工程目录myled，而myled.mcp 会出现在“D:\work\myled”目录中。

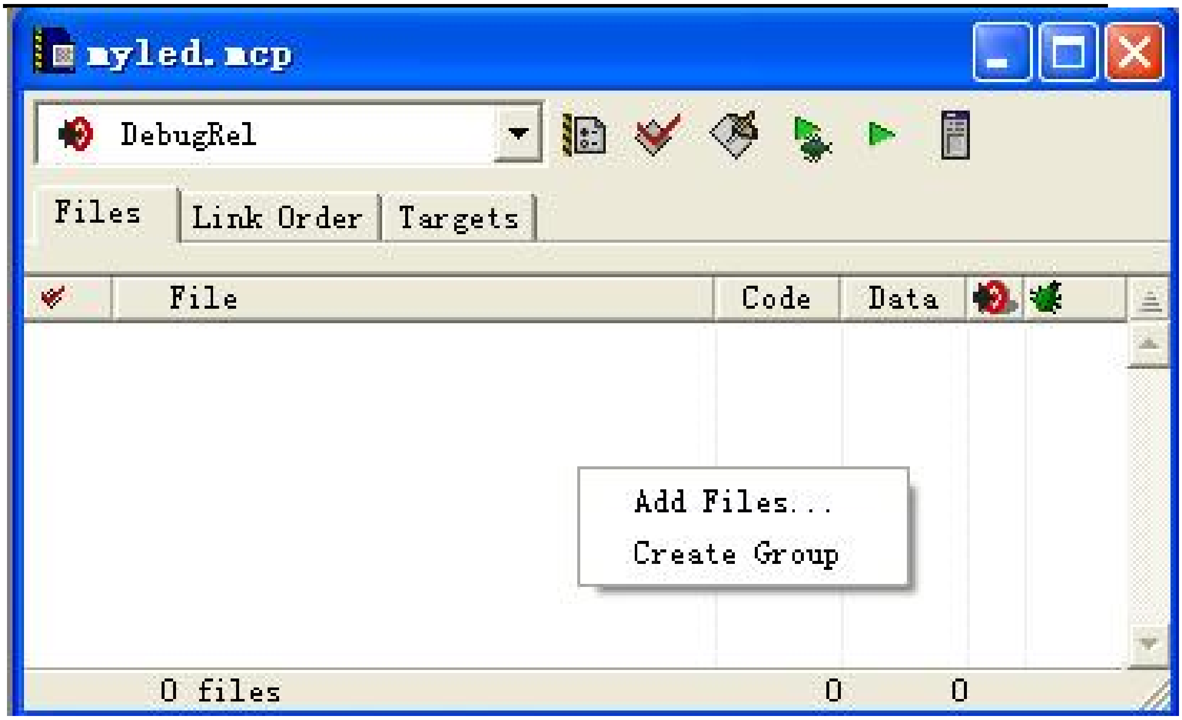


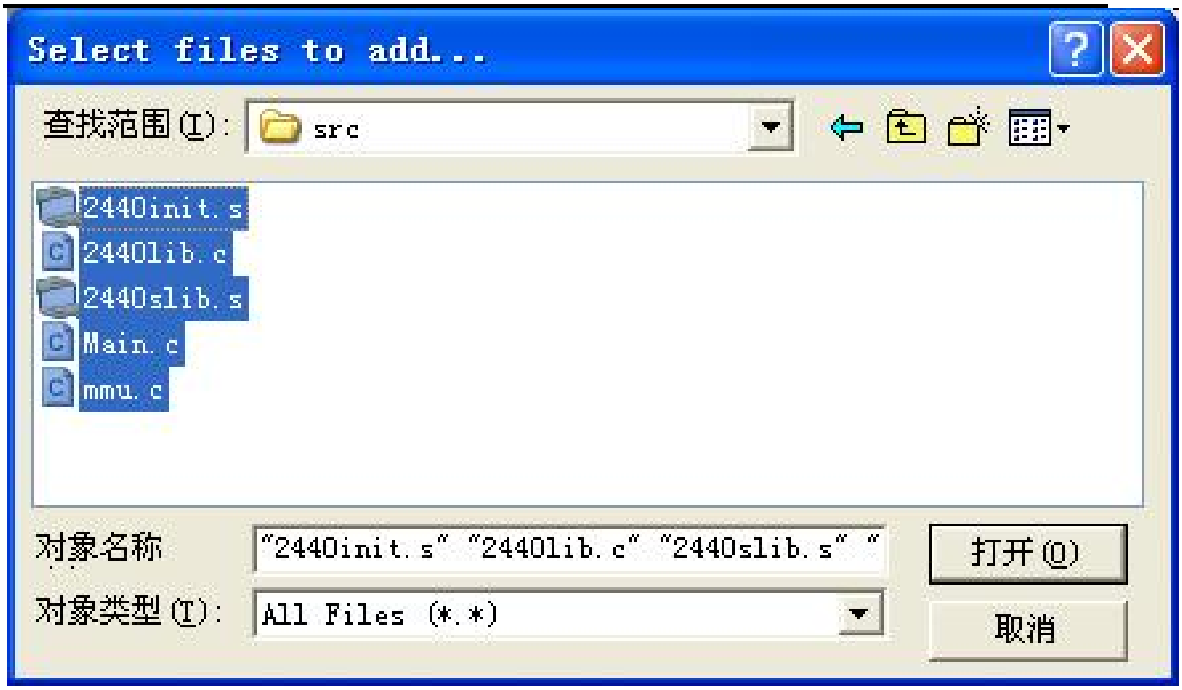


对于本例，我们将已经准备好的源文件及其目录（位于光盘的"非操作系统示例代码\myled"文件夹中）一起复制到myled 工程目录，如图



然后在 myled.mcp 项目窗口中，点鼠标右键或者ADS 菜单Project->Add Files…，如图开始添加该项目索需要的源代码。


myled.mcp
DebugRel
Files
Link Order
Targets
File
Code
Data
Add Files...
Create Group
0 files
0
0



点击“打开”按钮确定，这时会跳出如下图所示的提示选择窗口



这里请注意，我们在新建一个工程时，ADS 默认的target 时DebugRel，另外还有两个可用的target，分别为Realse 和Debug，它们的含义分别为：

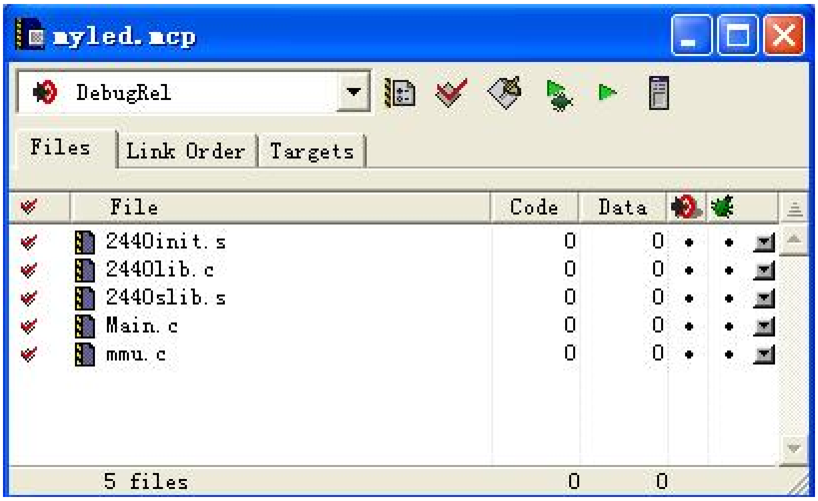
DebugRel: 使用该目标选项，在生成目标的时候，会为每一个源文件生成调试信息。

Debug: 使用该目标选项，在生成目标的时候，会为每一个源代码生成最完整的调试信息。

Release: 使用该目标选项，在生成目标的时候，不会生成任何调试信息。

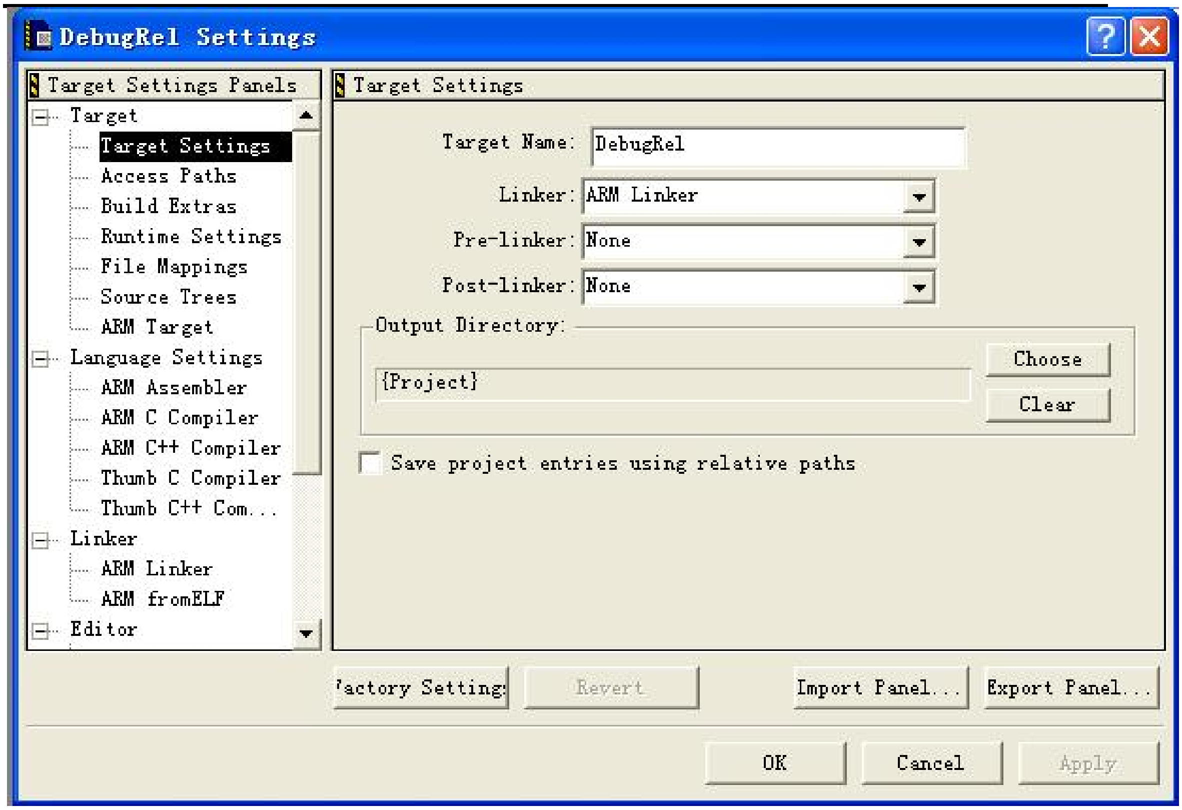
在本例中，我们使用默认的 DebugRel 选项。

到目前为止，一个完整的工程就已经建立了，下面我们开始对该工程进行编译和链接的配置。



4.1.2 建立一个工程

在进行编译和链接之前，首先需要对生成的目标进行配置，点【Edit】 菜单，选择【DebugRelSetting…】（注意：这个选项会因为用户选择的不同目标而有所不同），出现如图所示的设置窗口。



这里的设置有很多，我们主要介绍最常用的一些选项。

- **Target Setting**

Target Name 文本框显示了当前的目标设置。

Linker 选项为用户提供了要使用的连链接器，在这里选择默认的 ARM Linker，使用该链接器，将使用 armlink 链接编译器和汇编器生成相应的工程目标文件。

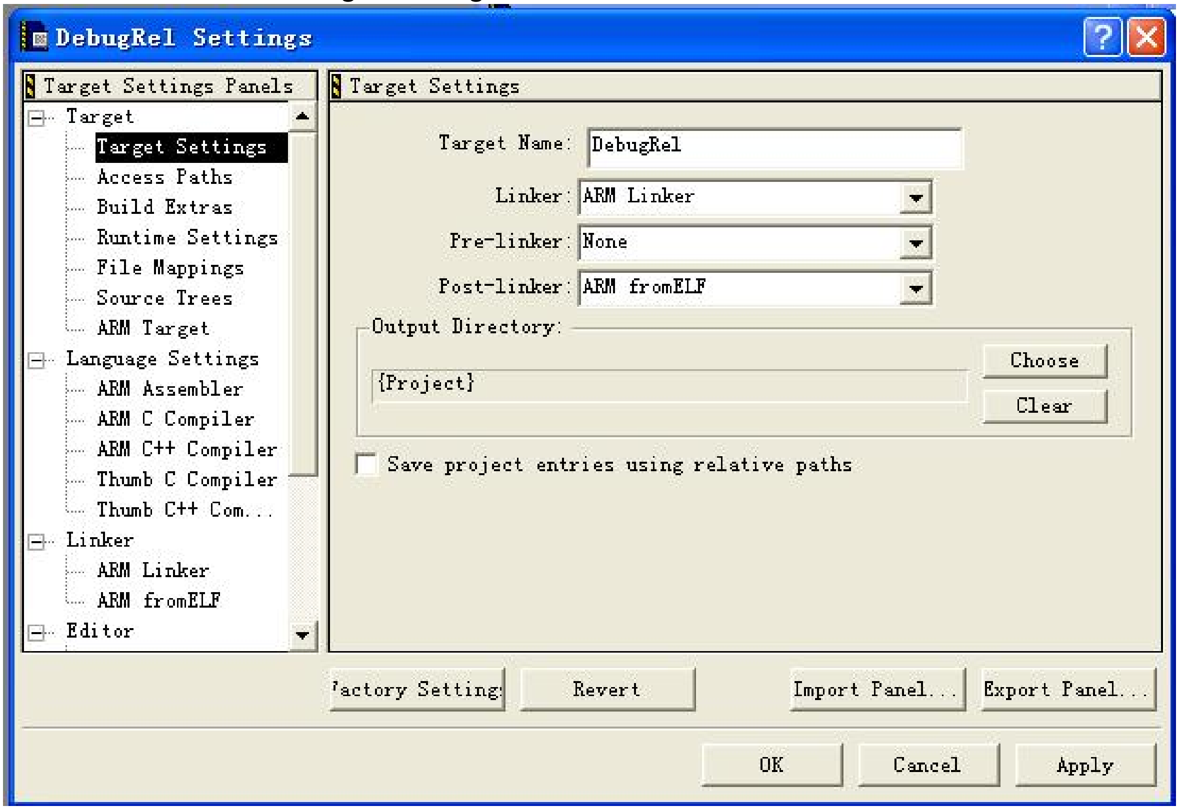
在 Linker 设置中，还有两个可选项，None 代表不对生成的各个源代码目标文件进行链接，ARM Librarian 表示将编译或者汇编得到的目标文件转换为 ARM 库文件，对于本例，使用默认的链接器 ARM Linker。

Pre-Linker: 目前 ADS 并不支持该选项。

Post-Linker: 选择在链接完成后，还要对输出文件进行的操作。因为在本例中，希望生成一个可以烧写到 Flash 中去的二进制代码，所以在此选择 ARM fromELF，表示在链

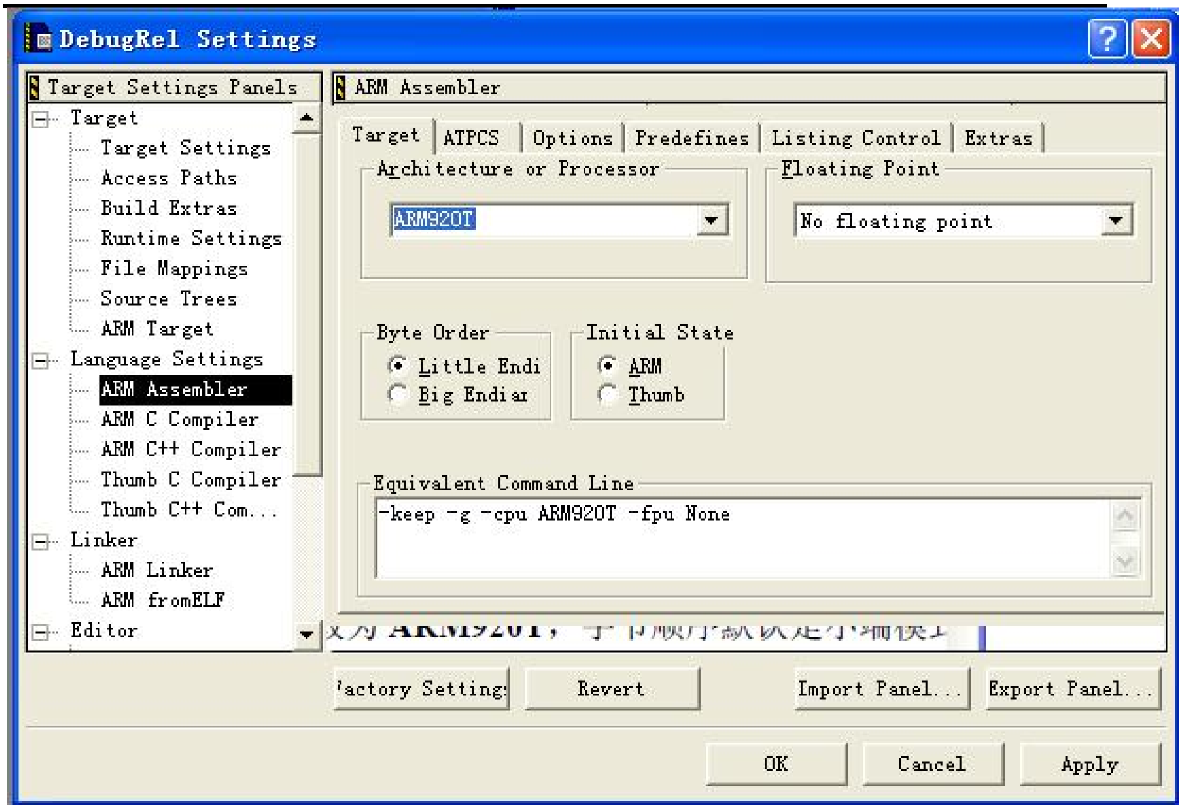
接生成映象文件后，再调用 fromELF 命令将含有调试信息的 ELF 格式的映象文件转换为其他格式的文件。

Target Setting 选择最后设置如图所示：

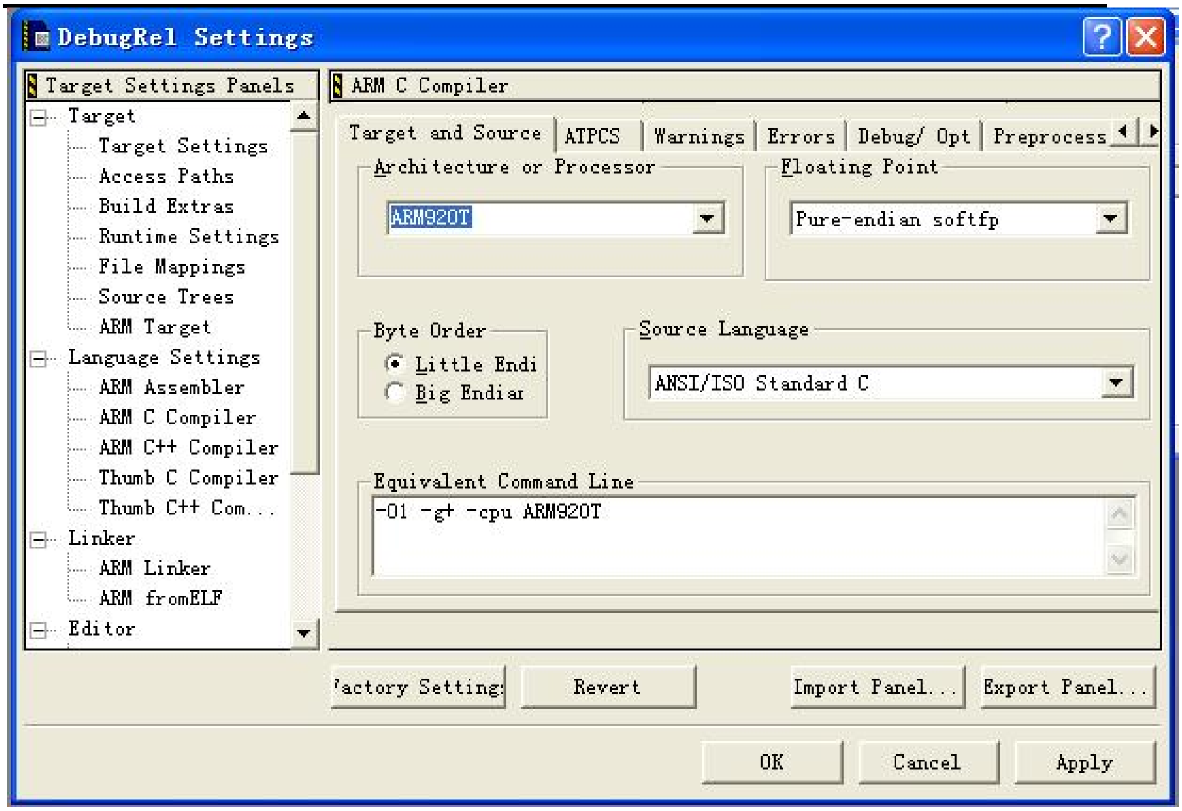


l　Language Settings

因为在本例中包含汇编代码，所有要用到汇编器，点接选择 ARM Assembler，在右侧出现相应的设置选项，在 ADS 集成开发环境中用的汇编器是 armasm，默认的 ARM 体系结构是 ARM7TDMI，在此要改为 ARM920T，字节顺序默认是小端模式，其他设置，采用默认值即可，如图所示：



本例中还包含了 C 语言代码,因此还需要设置 ARM C Compiler 选项,点接选择 ARM C Compiler,在右侧出现相应的设置选项,在 ADS 集成开发环境中用的汇编器是 armcc,默认的 ARM 体系结构是 ARM7TDMI,在此要改为 ARM920T,字节顺序默认是小端模式,其他设置,采用默认值即可,如图所示:



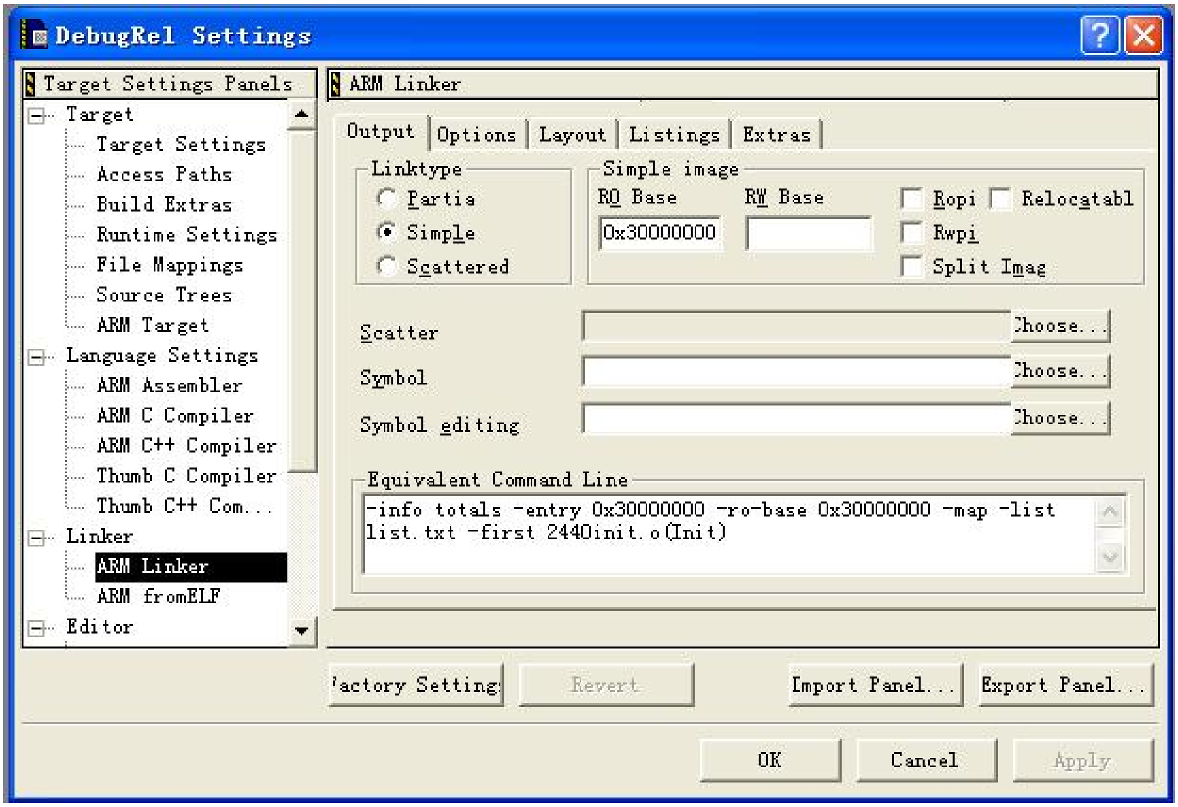
细心的读者可能会注意到，在设置框的右下脚，当对某项设置进行了修改，该行中的某个选项就会发生相应的改动，实际上，这行文字显示的就是相应的编译或者链接选项，由于有了 CodeWarrior，开发人员可以不用再去产科繁多的命令行选项，只要在界面中选择或者撤销某个选项，软件就会自动生成相应的代码，该命令框为习惯在 DOS 下键入命令行的用户提供了极大的方便。

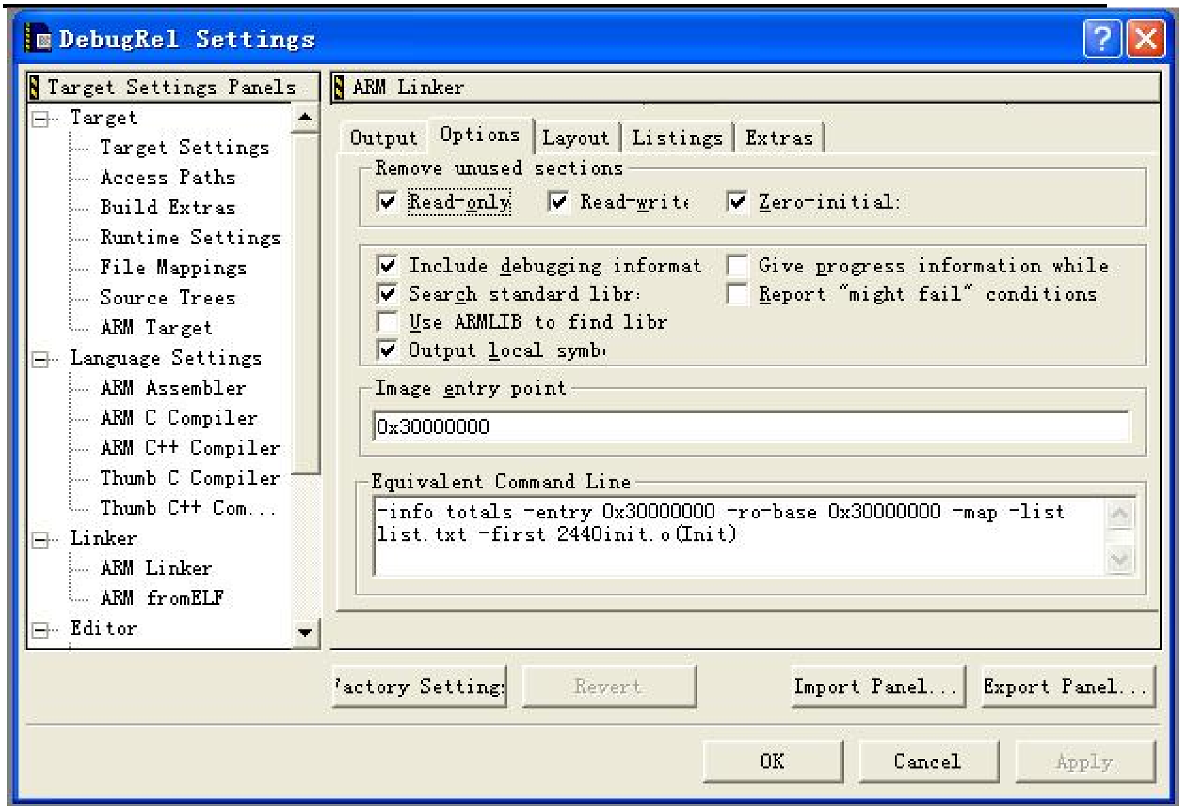
l Linker 设置

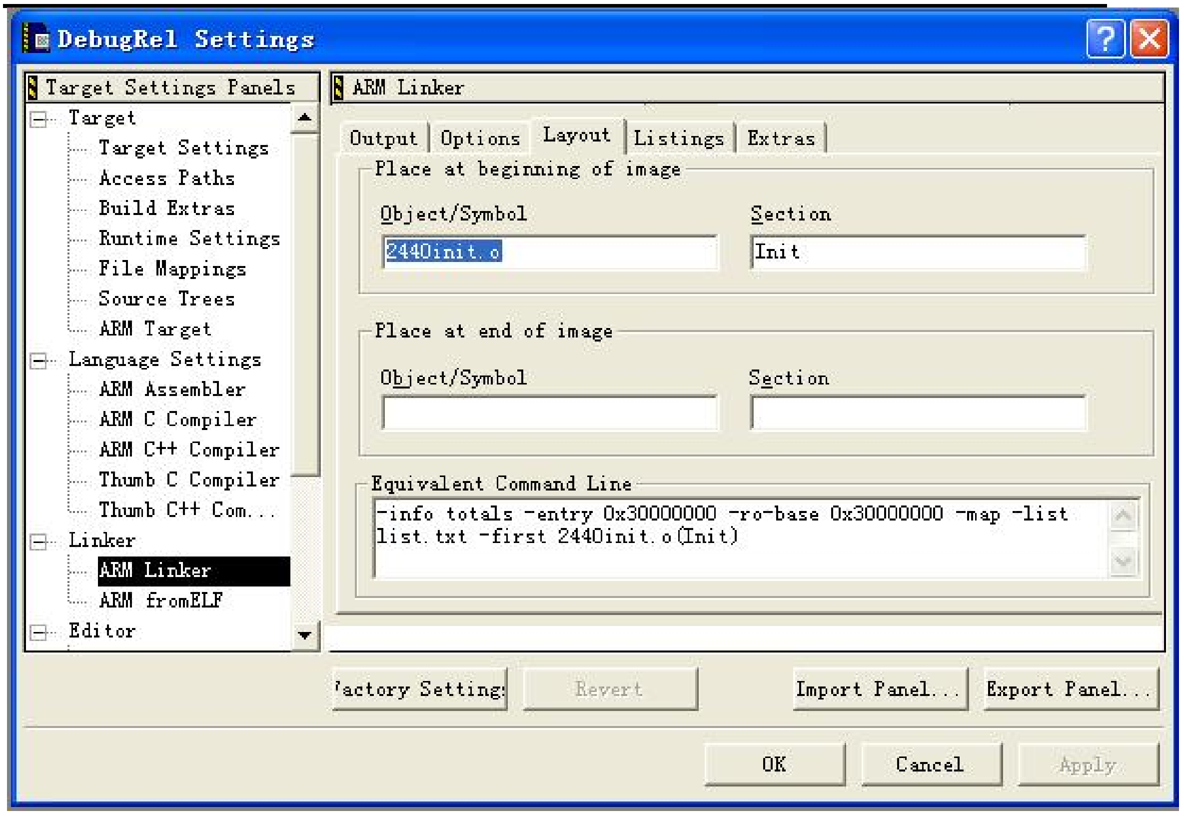
提示：如果您对 ADS 设置不熟悉，可以使用我们缺省的项目文件，以下部分仅供参考。

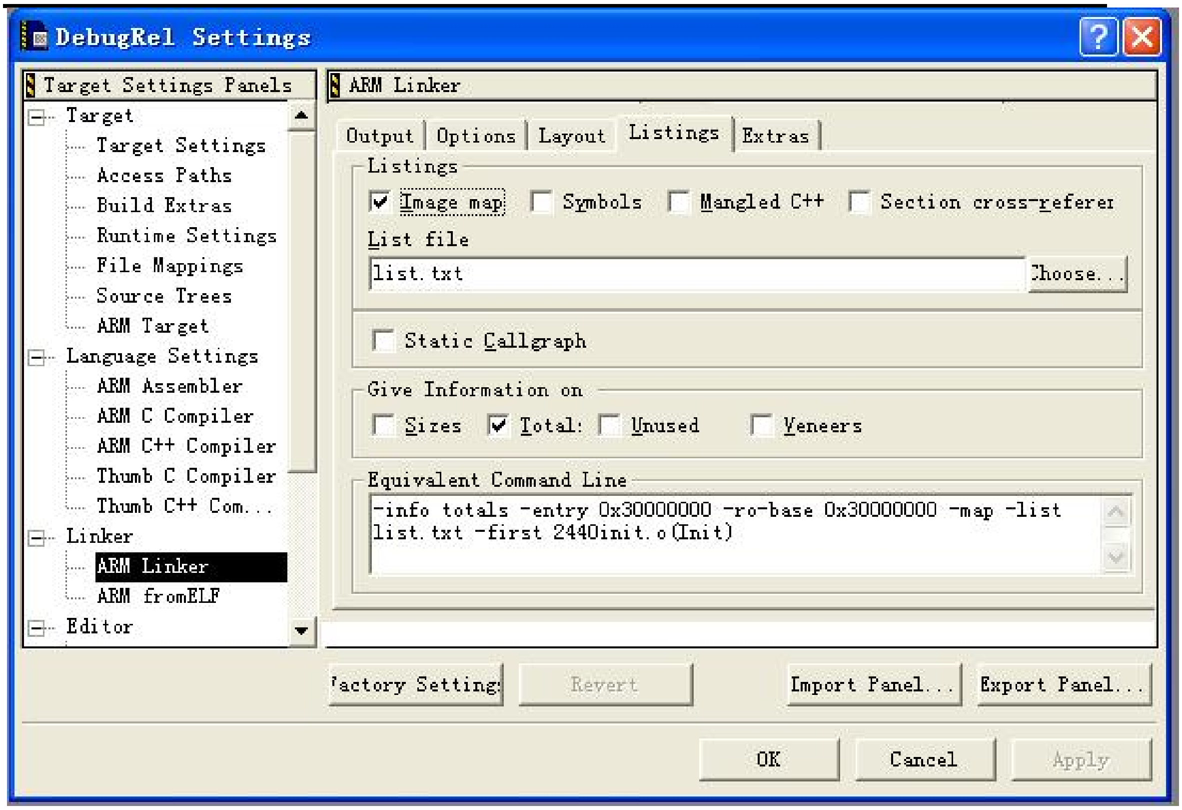
点接选择 ARM Linker，在右侧出现相应的设置选项，我们在此详细介绍这些设置框，因为这些选项对最终生成的文件有着直接的影响。

在标签 Output 中，Linktype 中提供了三种链接方式。Partial 方式表示链接器只进行部分链接，经过部分链接生成的目标文件，可以作为以后进一步链接时的输入文件。Simple 方式是默认的链接方式，也是最为频繁使用的链接方式，它链接生成简单的 ELF 格式的目标文件，使用的是链接器中指定的地址映象方式。Scattered 方式使得链接器要根据 scatter 格式文件指定的地址映象，生成复杂的 ELF 格式的映象文件，这个选项一般很少用到。




DebugRel Settings
Target Settings Panels
Target
Target Settings
Access Paths
Build Extras
Runtime Settings
File Mappings
Source Trees
ARM Target
Language Settings
ARM Assembler
ARM C Compiler
ARM C++ Compiler
Thumb C Compiler
Thumb C++ Com...
Linker
ARM Linker
ARM fromELF
Editor
ARM Linker
Output
Options
Layout
Listings
Extras
Remove unused sections
Read-only
Read-write
Zero-initial:
Include debugging informat
Give progress information while
Search standard libr:
Report "might fail" conditions
Use ARMLIB to find libr
Output local symb
Image entry point
0x30000000
Equivalent Command Line
-info totals -entry 0x30000000 -ro-base 0x30000000 -map -list
list.txt -first 2440init.o(Init)
'actory Setting:
Revert
Import Panel...
Export Panel...
OK
Cancel
Apply


DebugRel Settings
Target Settings Panels
Target
Target Settings
Access Paths
Build Extras
Runtime Settings
File Mappings
Source Trees
ARM Target
Language Settings
ARM Assembler
ARM C Compiler
ARM C++ Compiler
Thumb C Compiler
Thumb C++ Com...
Linker
ARM Linker
ARM fromELF
Editor
ARM Linker
Output
Options
Layout
Listings
Extras
Place at beginning of image
Object/Symbol
Section
2440init.o
Init
Place at end of image
Object/Symbol
Section
Equivalent Command Line
-info totals -entry 0x30000000 -ro-base 0x30000000 -map -list list.txt -first 2440init.o(Init)
'actory Setting:
Revert
Import Panel...
Export Panel...
OK
Cancel
Apply



在本例中，我们选择使用 Simple 方式，这里有一些设置：

RO Base: 这个文本框设置包含 RO 段的加载域和运行域为同一个地址，默认是 0x8000。这里用户要根据自己的硬件实际 SDRAM 地址空间来修改这个地址，保证这里填写的地址，是程序运行时，SDRAM 地址空间所能到达的范围，针对本目标板，SDRAM 的空间范围是 0x3000000-0x34000000，因此这里设置为 0x30000000。

RW Base: 这个文本框设置了包含 RW 和 ZI 输出段的运行域地址。如果选中 split 选项，链接器生成的映象文件将包含两个加载域和两个运行域，此时在 RW Base 中所输入的地址为包含 RW 和 ZI 输出段的域设置了加载域和运行域地址。

Ropi: 选中这个设置将告诉链接器使包含有 RO 输出段的运行域位置无关。使用这个选项，链接器将保证下面的操作：检查各段时间的重寻址是否有效；确保任何由 armlink 自身生成的代码是只读位置无关的。

Rwpi: 选中该选项将会告诉链接器使包含 RW 和 ZI 输出段的运行域无关。如果这个选项没有被选中，域就标识为绝对。每一个可写的输入段必须是和读写位置无关的。如果这个选项被选中，链接器将进行下面的操作:

检查可读/写属性的运行域的输入段是否设置了位置无关属性;

检查在各段之间的重地址是否有效;

在 Region$$Table 和 ZISection$$Table 中添加基于静态存储器 sb 的选项。

该选项要求 RW Base 有值，如果没有给他指定数值的话，默认为 0。

Split Image: 选择这个选项把包含 RO 和 RW 的输出段的加载域分成 2 个加载域，一个是包含 RO 输出段的域，一个是包含 RW 输出段的域。

这个选项要求 RW Base 有值，如果没有给 RW Base 选项的值，则默认是 0。

Relocatable: 选择这个选项保留了映像文件的重寻址偏移量。这些偏移量为程序加载器提供了有用信息。在 Options 选项中，需要读者引起主义的是 Image entry point 文本框。

它指定映像文件的初始入口点地址值，当映像文件被加载程序加载时，加载程序会跳转到该地址处执行。如果需要，用户可以在这个文本框中输入下面格式的入口点:

入口点地址：这是一个数值，例如 -entry 0x0

符号：该选项指定映像文件的入口点为该符号所代表的地址处，比如：-entry int_handler ，如果该符号有多处定义存在，armlink 将产生出错信息。

offset+object(section): 该选项指定在某个目标文件的段的内部的某个偏移量处为映像

文件的入口地址，例如：-entry8+startup(startuoseg)在此处指定的入口点用于设置 ELF 映像文件的入口地址。需要引起注意的是，这里不可以用符号 main 作为入口点地址符号，否则将会出现类似“Image dose not have an entry point(Not specified or not set due to multiple choice)”的错误信息。在 Layout 选项中，需要设置 2440init.o 目标文件中的 Init 为整个文件的入口点。关于 ARM Linker 的设置还很多，对于想进一步深入了解的读者，可以查看帮助文件，都有很详细的介绍。

在 Linker 下还有一个 ARM fromELF:

fromELF 是一个实用工具，它实现**将链接器，编译器和汇编器的输出代码进行格式转换的功能**。例如，将 ELF 格式的可执行映像文件转换成可以烧写到 ROM 的二进制格式文件；对输出文件进行反汇编，从而提取出有关目标文件的大小，符号和字符串表以及重寻址等信息。只有在 Target 设置中选择了 Post-linker，才金可以使用该选项。

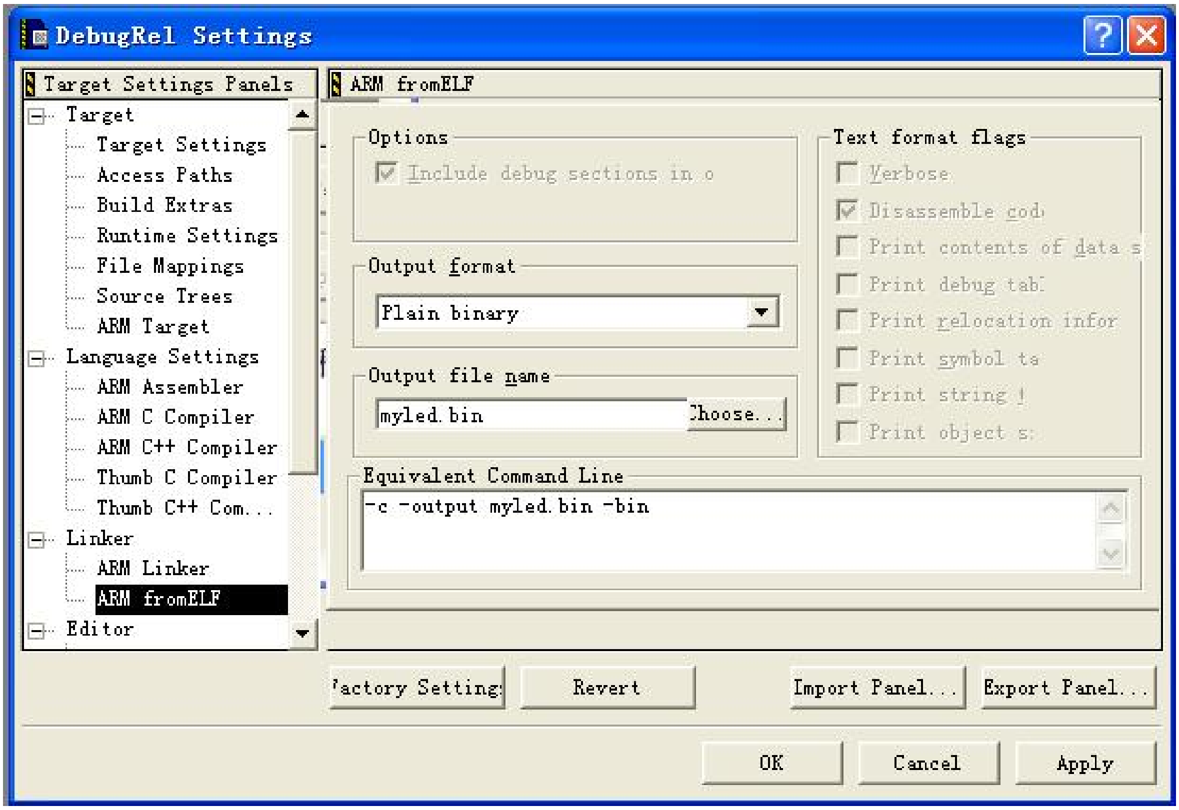
在 Output format 下拉框中，为用户提供了多种可以转换的目标格式，本例选择 Plain

binary，这是一个二进制格式的可执行文件，可以被烧写到目标板的 Flash 中。

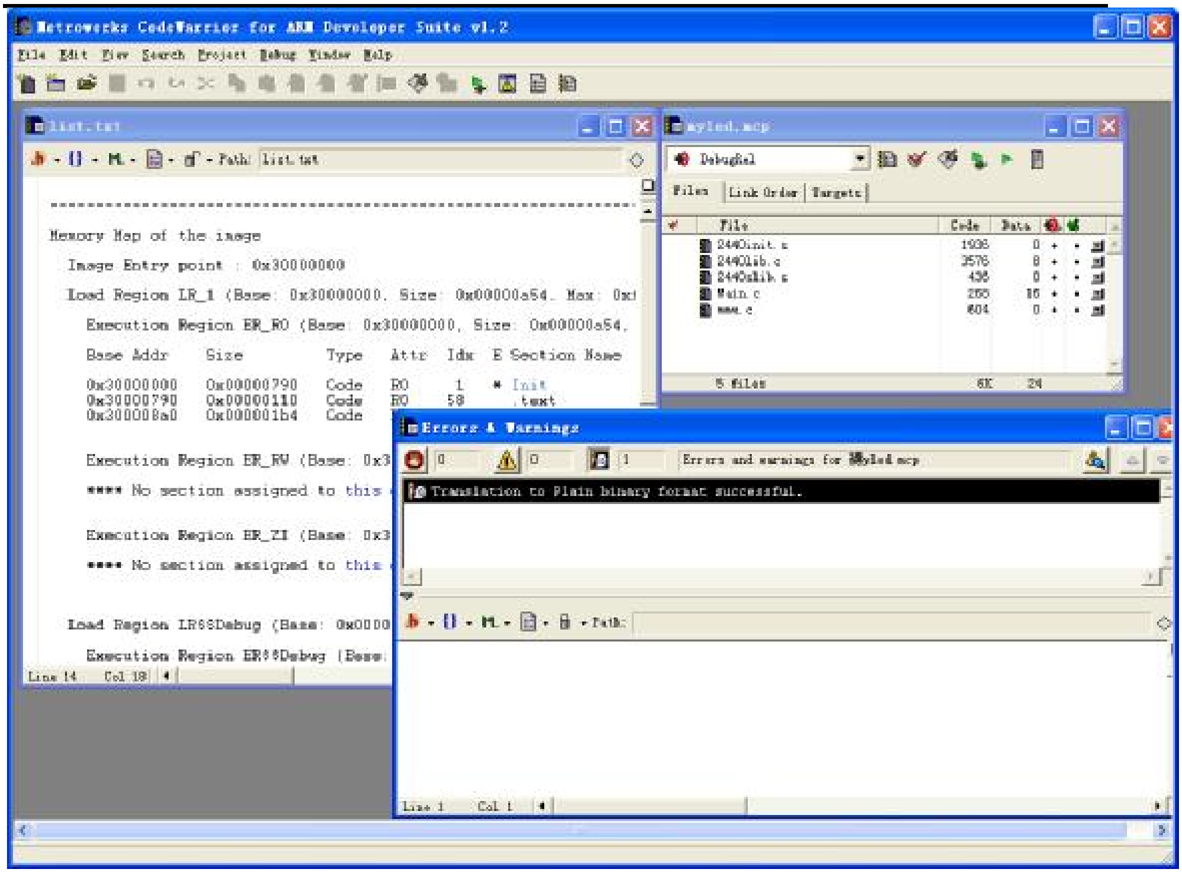
在 Output file name 文本域输入期望生成的输出文件存放的路径，或通过点击 Choose...按钮从文件对话框中选择输出文件。如果在这个文本域不输入路径名，则生成的二进制文件存放在工程所在的目录下。进行好这些相关的设置后，以后在对工程进行 make 的时候，

CodeWarrior IDE 就会在链接完成后调用 fromELF 来处理生成的映像文件。

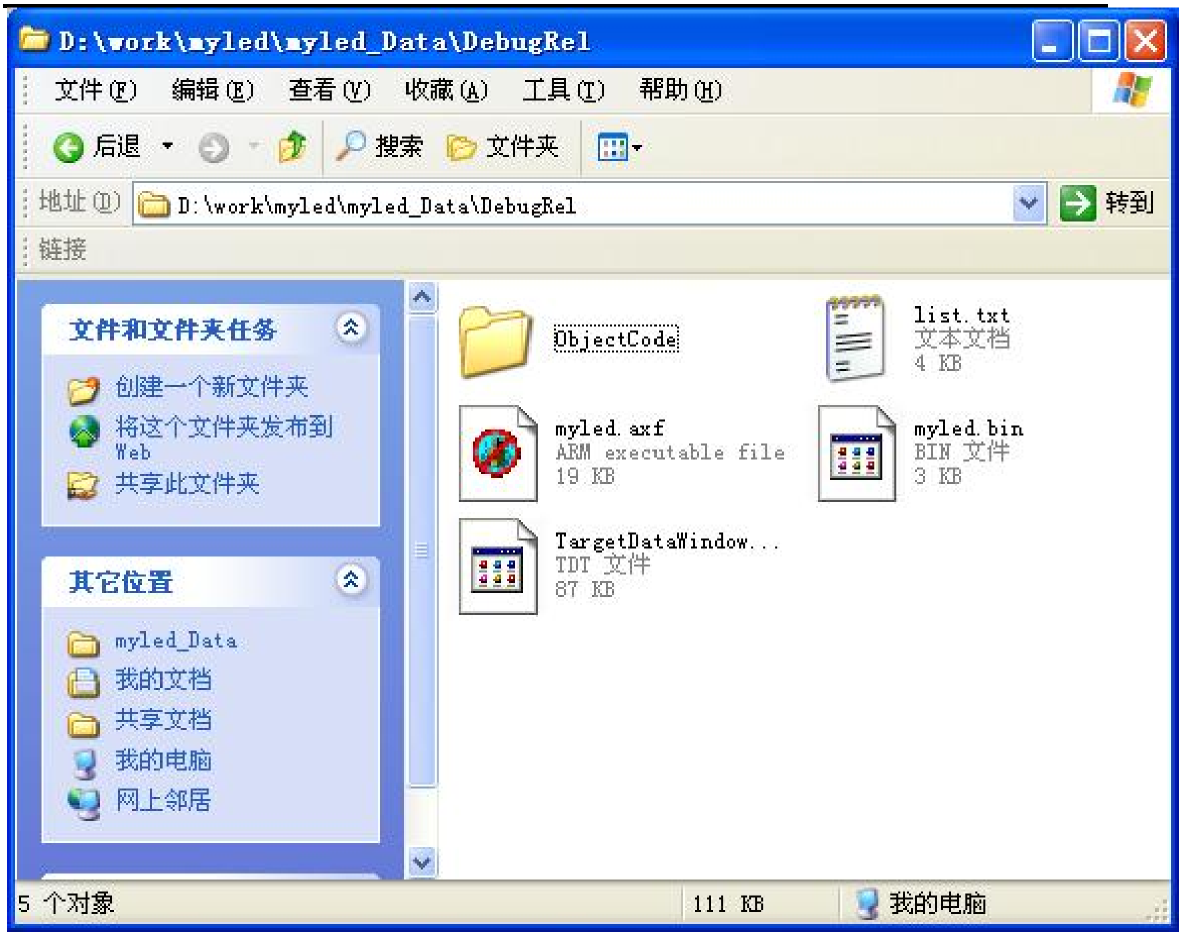
对于本例的工程而言，到此，就完成了 make 之前的设置工作了。



点击 CodeWarrior IDE 的菜单 Project 下的 make 菜单，就可以对工程进行编译和链接了。编译链接结果如图：



最后在“D:\work\myled\myled_Data\DebugRel”目录下生成 myled.bin，同时还有 myled.axf 文件，它是用于调试的，如图：



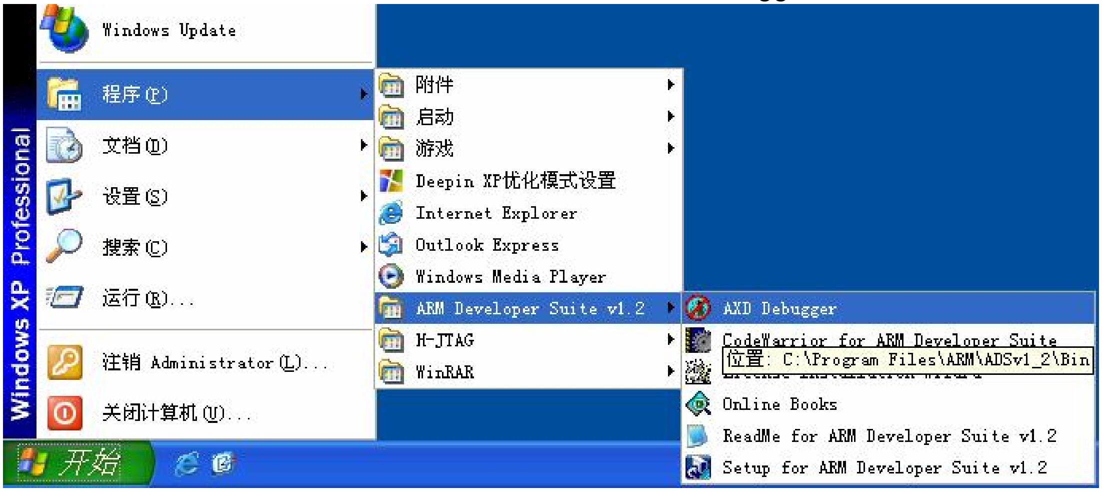
4.2 使用 HJTAG 进行代码调试

在光盘目录**\windows 平台工具\H-JTAG** 中，找到 H-JTAG_V0.9.2.rar，解压后点击 H-JTAG.exe 进行安装。

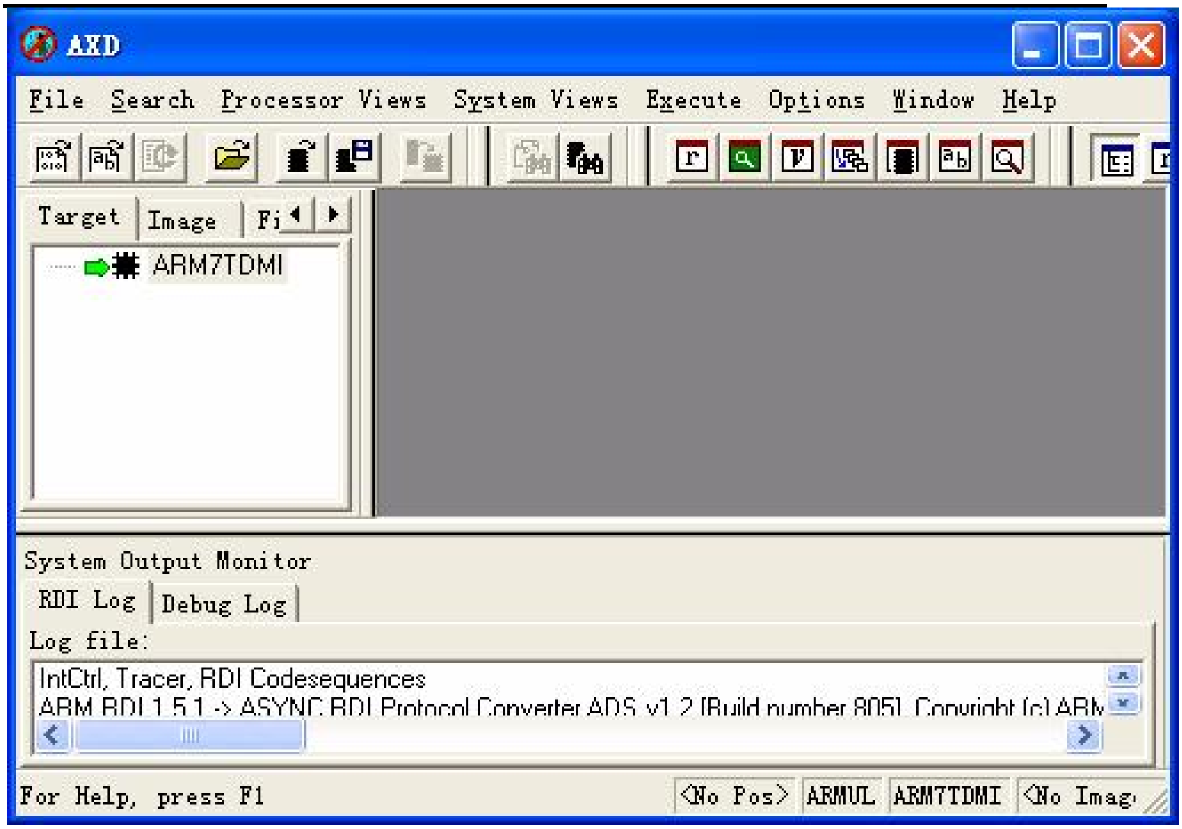
4.2.1 为 HJTAG 配置 AXD DEBUGGER

(1)运行 AXD Debugger

如图所示打开运行 ADS1.2 软件的调试软件－AXD Debugger:

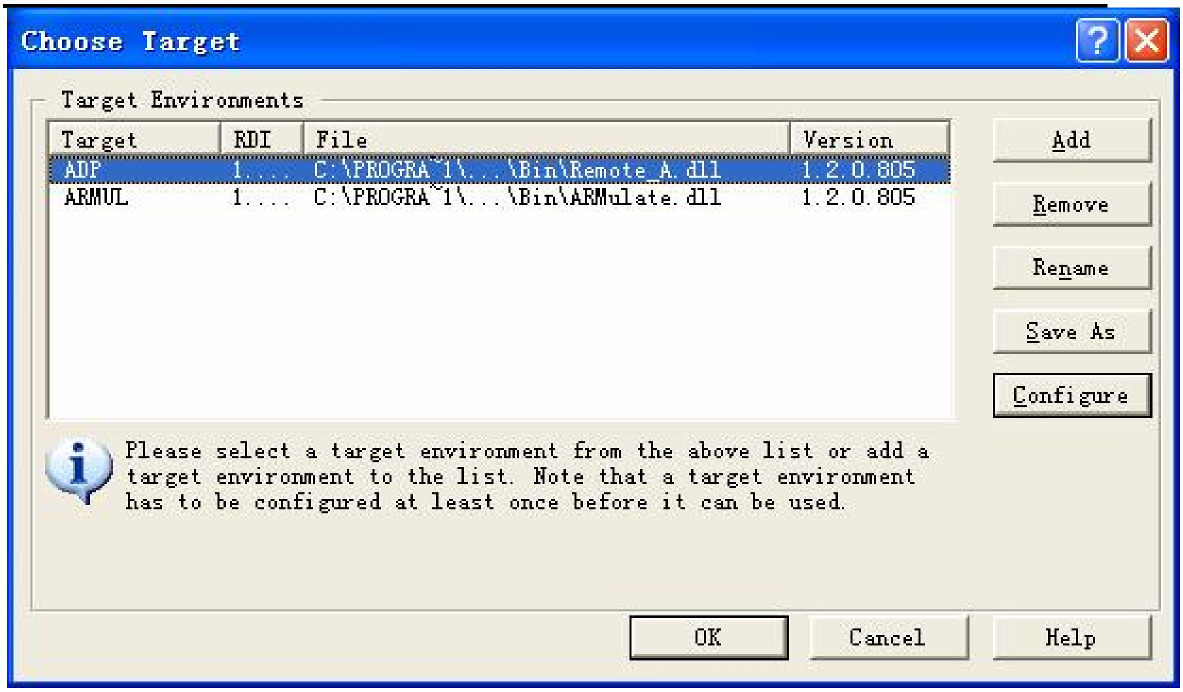


AXD Debugger 主界面:

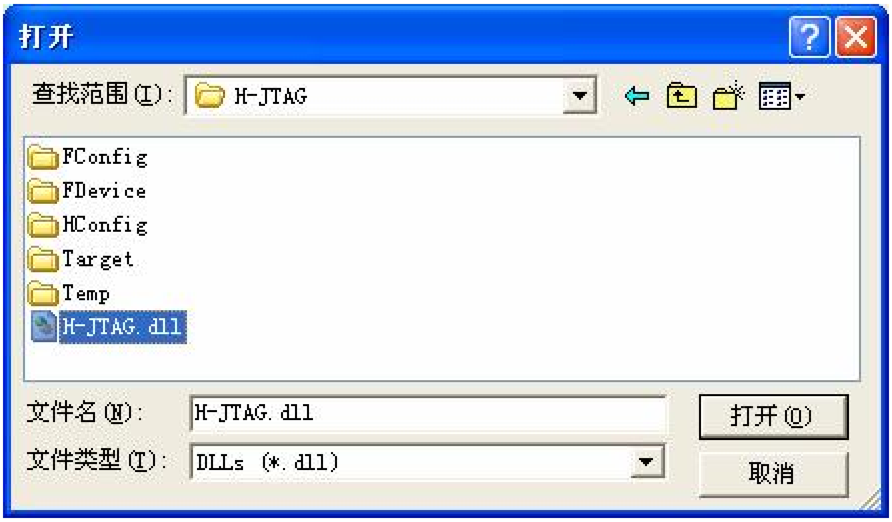


(2)设置 AXD Debugger

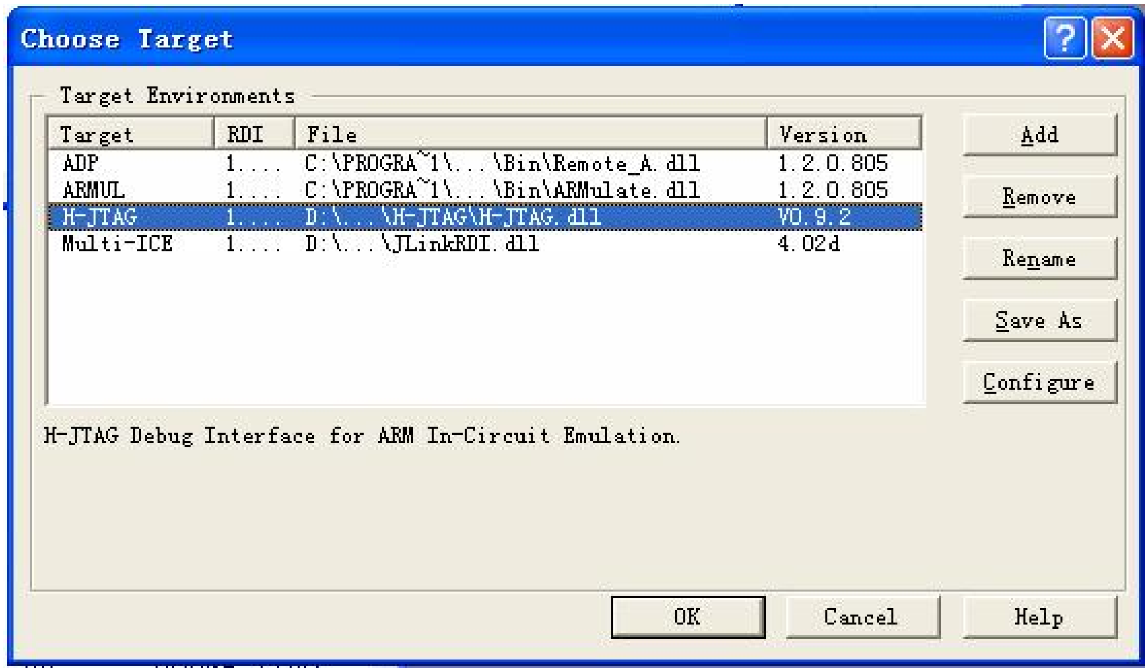
点菜单【Options】|【Confiuguer Target…】，出现如下窗口：



在该窗口中点击 Add 按钮，跳出选择文件对话框，找到 H-JTAG 安装目录，选择并打开里面的 H-JTAG.dll 文件，如图。



此时会在 Choose Target 窗口中多了一项 H-JTAG，如图，点 OK 返回 AXD Debugger 主界面。



4.2.2 使用 HJTAG 在 ADS1.2 环境下进行仿真调试

关闭并重新启动 AXD Debugger，点击菜单 File->Load Image，找到您想调试的调试目
标文件(*.axf 格式)，如我们前面编译生成的 myled.axf 文件，打开它就自动启动目标映象通过 Jtag 下载至目标板，这时在 AXD Debugger 底部的状态栏会出现下载过程提示，下载完毕就可以进行单步或者全速调试了，调试过程中您可以看到 CPU 各个寄存器的值，也可以设置断点等，详细的用法请参考 ADS 附带的英文说明手册，这基本上和常见的 VisualC++等集成开发环境是类似的。

4.3 编译运行烧写 2440test

2440test 是源自三星的一个非操作系统测试程序，里面集成了很多小型的测试程序，涉及到 GPIO 的配置，中断的编写，常见接口的测试使用等，其中每一部分的测试代码都有很强的独立性，非常适合“ARM 基础性”练习实验。2440test 是基于 ADS1.2 开发环境创建的，它编译出的二进制文件不能下载到 Linux 或者 wince 系统中运行，只能下载到内存指定地址（这里是 0x30000000）运行，也可以烧写到 Nand Flash 中运行。

2440test 的代码仅供学习参考使用，它在某些方面可能是不完善的，我们不对它的原理做任何解释说明，也没有计划去深入完善它。

注意：经改进，2440test 目前可以烧写到 64M/128M-1GB Nand Flash 版的 GQ2440 中，并自动适应运行。

本开发板提供的 2440test 对于开机后 LCD 显示的支持，可以通过 \2440test\inc\LCD_TFT.h 中的 LCD_TYPE 定义来选择(通过去掉前面的注释符”//”可以选择你需要的 LCD 型号)：

```
#define LCDW35      1    // 东华屏(320X240)
#define LCDS35      2    // 三星屏(320X240)
#define LCDT35      3    // 东芝屏(240X320)
#define LCDW43      4    // 东华 4.3 寸屏
#define LCD57       5    // 5.7 寸屏
#define LCDA70      6    // 7.0 寸屏
#define VGA         7    //VGA
#define LCDA104     8    //10.4 寸屏

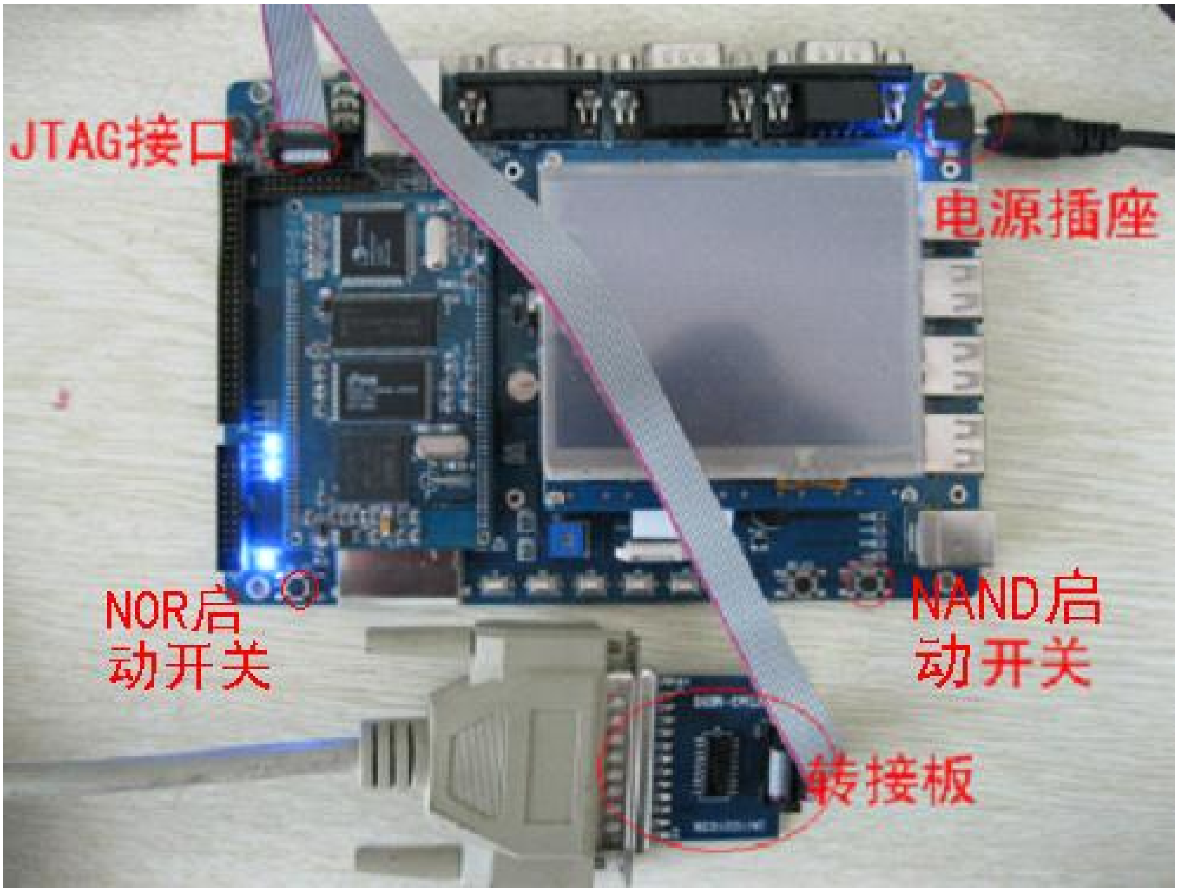
#define LCD_Type    LCDS35    //设定屏的类型
```

光盘中默认的是 LCDS35

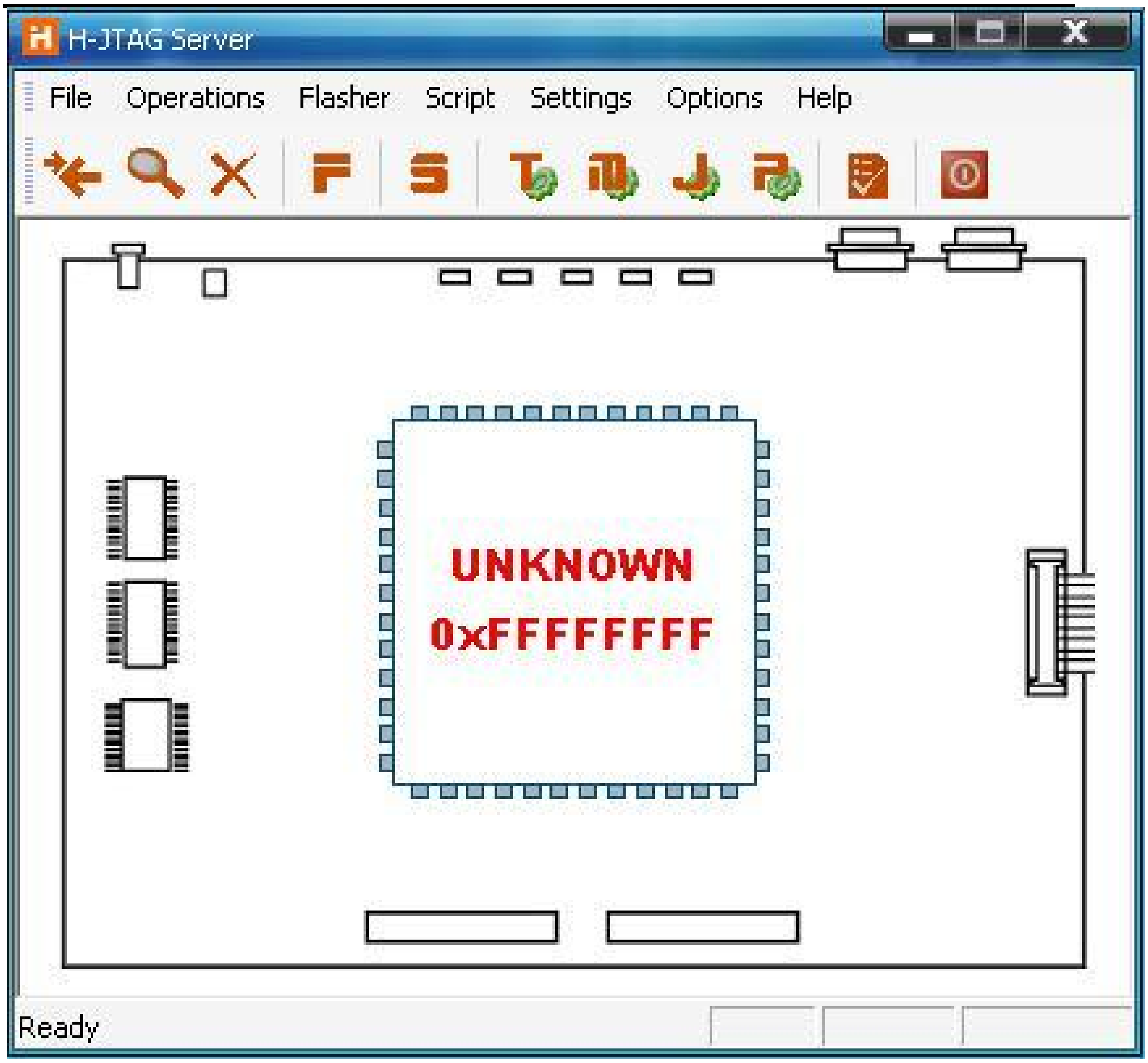
下面是 2440test 的编译、运行和烧写的方法，代码的原理性请用户自行理解。

4.3.1 编译和使用 HJTAG 调试 2440test

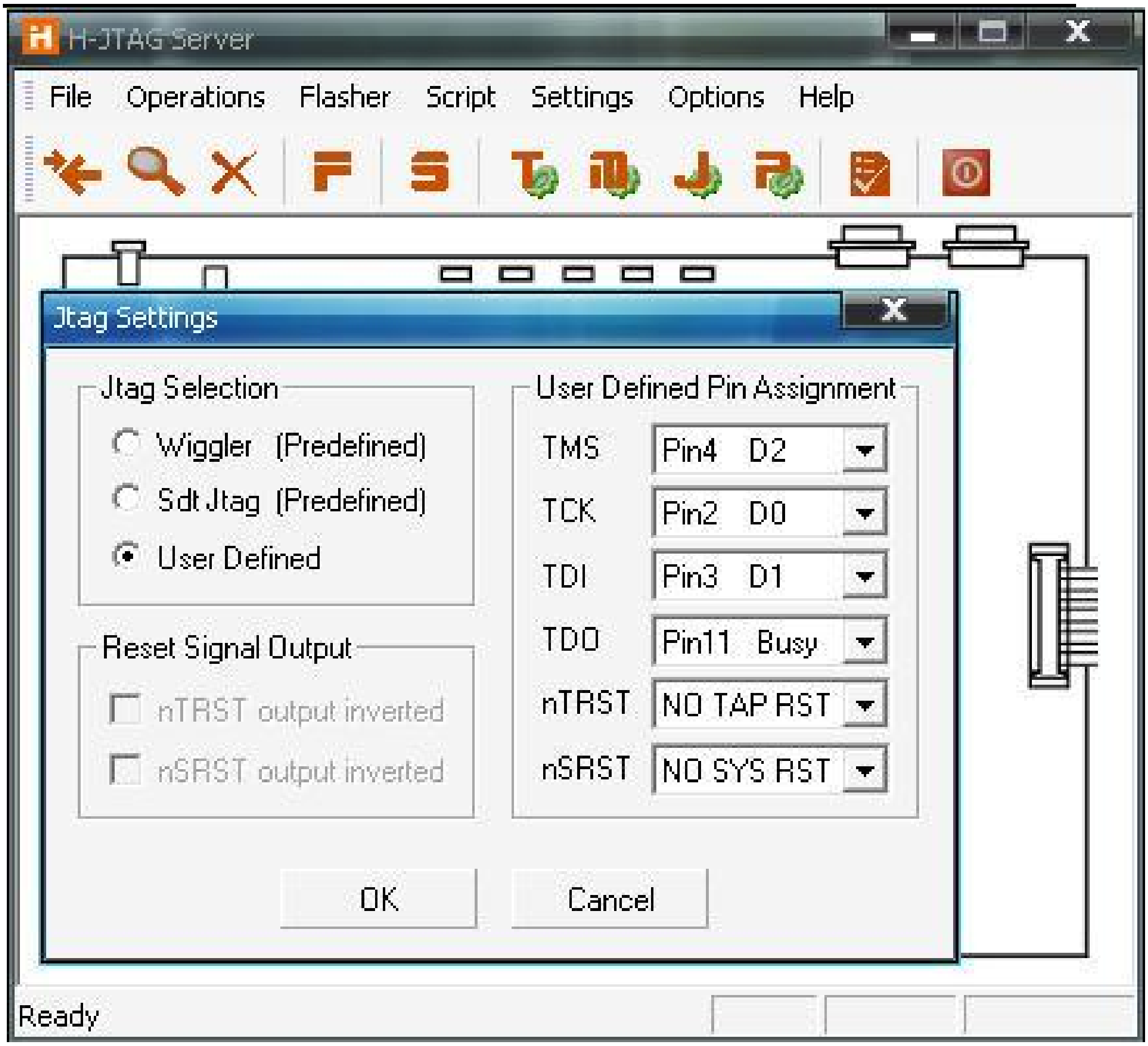
使用开发板附带的 JTAG 小板连接开发板的 JTAG 接口，使用附带的串口线连接开发板的串口。连接方式如图：



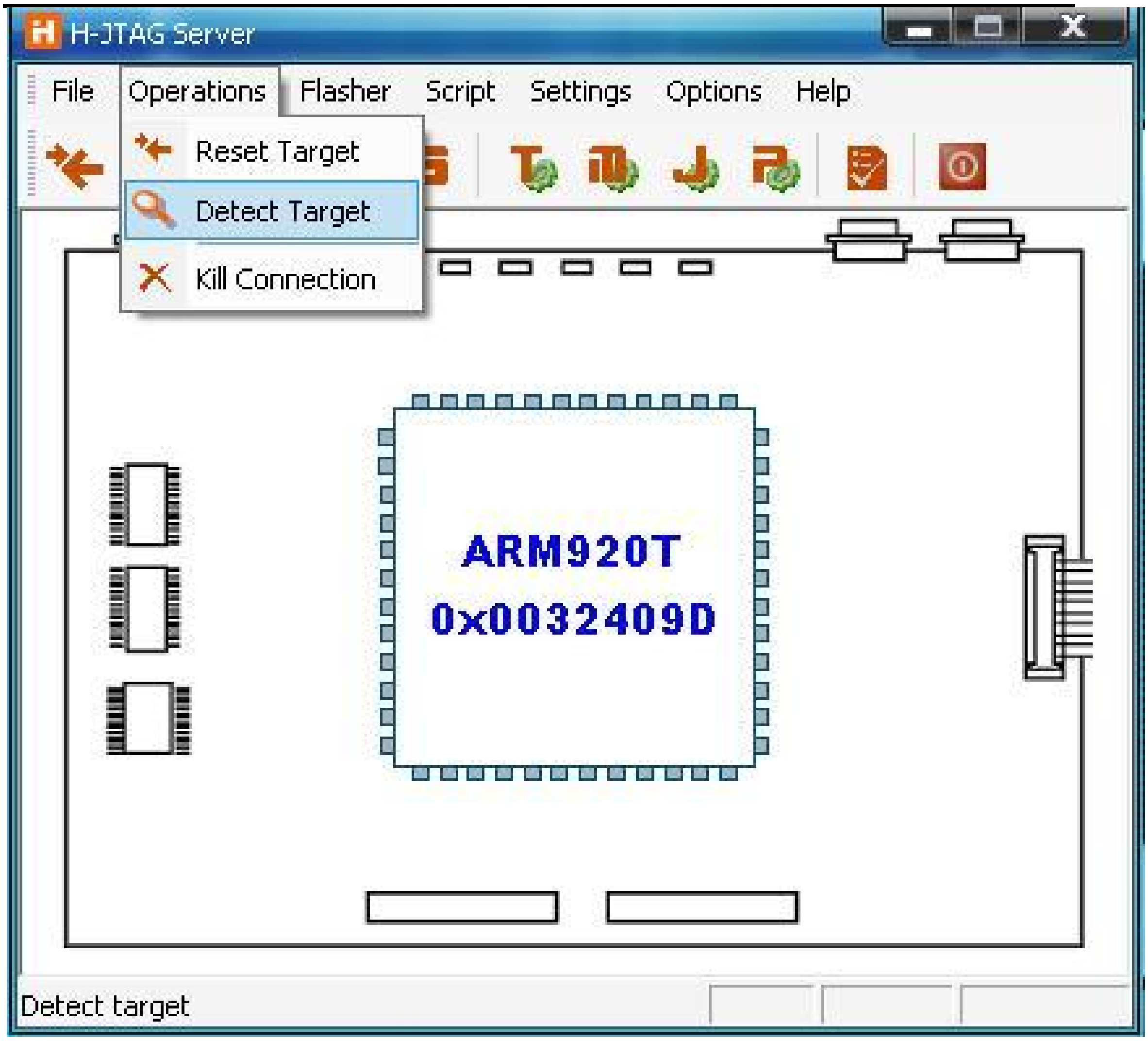
打开开发板电源；运行 H-JTAG 程序，这时 H-JTAG 还无法识别到目标开发板。



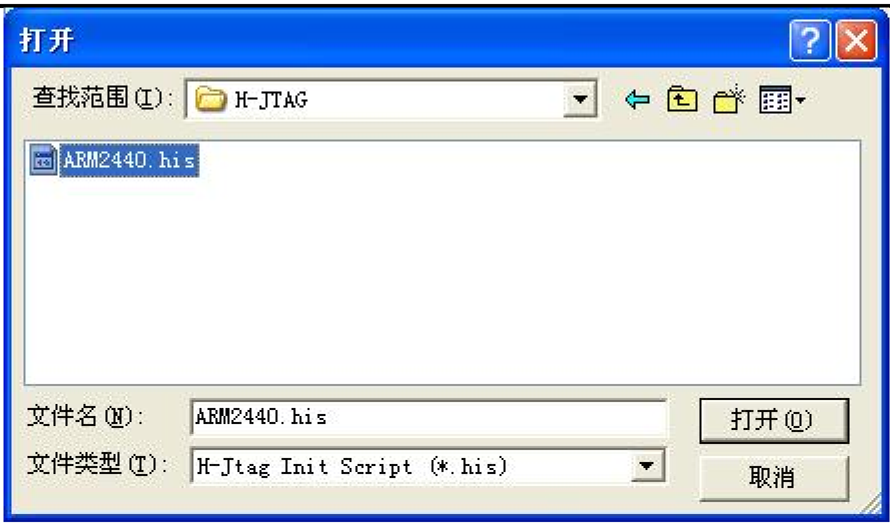
点击图标，或者通过下拉菜单【Settings】|【Jtag Settings】进行 Jtag 端口配置。配置方法如图所示:



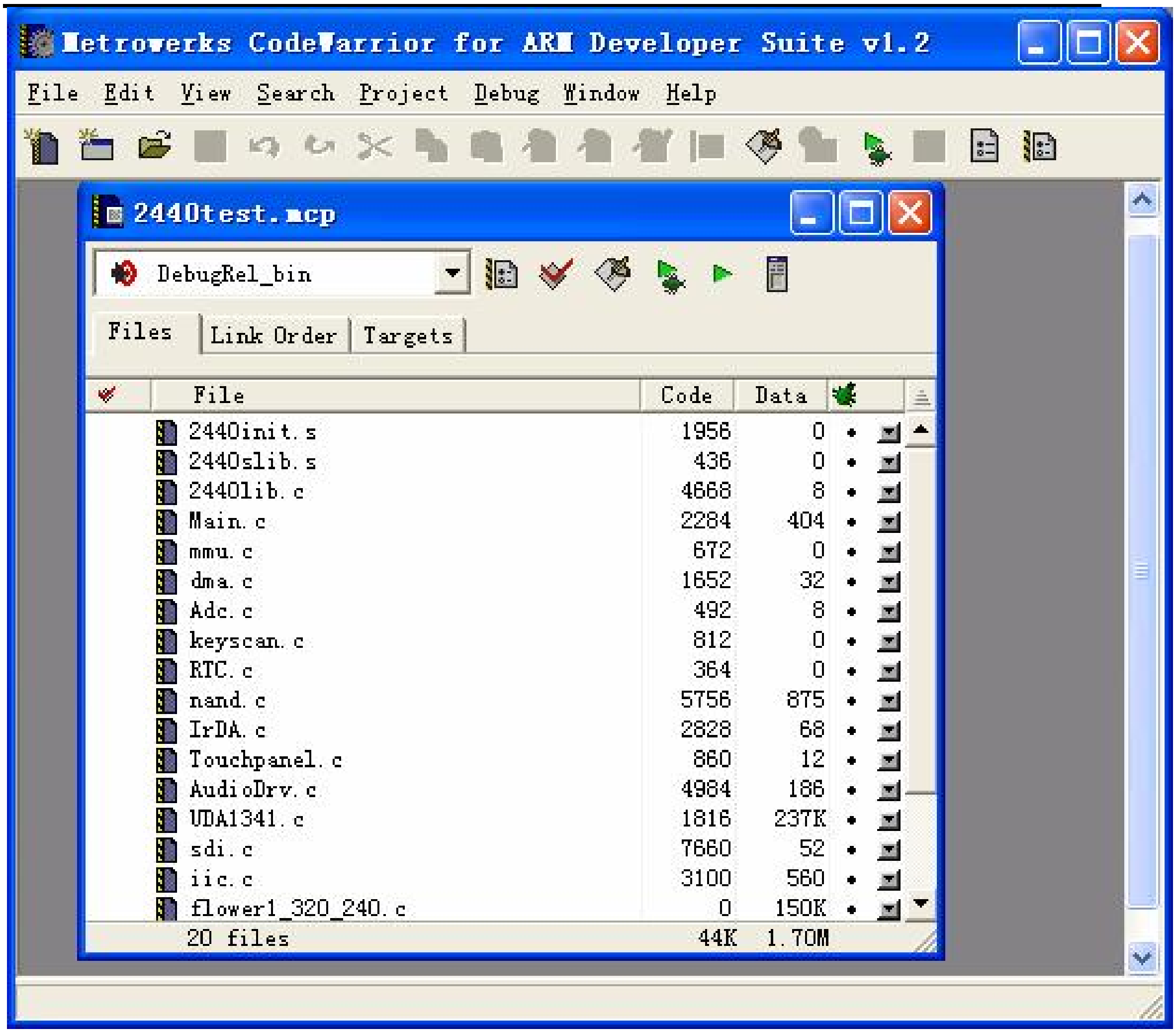
点击图标，或者通过下拉菜单【Operations】|【Detect Target】检测设备。如图所示，可以看到 H-JTAG 成功识别到目标板处理器。



点击图标，或者通过下拉菜单【Script】|【Init Script】添加初始化脚本。方法如图所示。脚本位于光盘目录\windows 平台工具\H-JTAG 下。

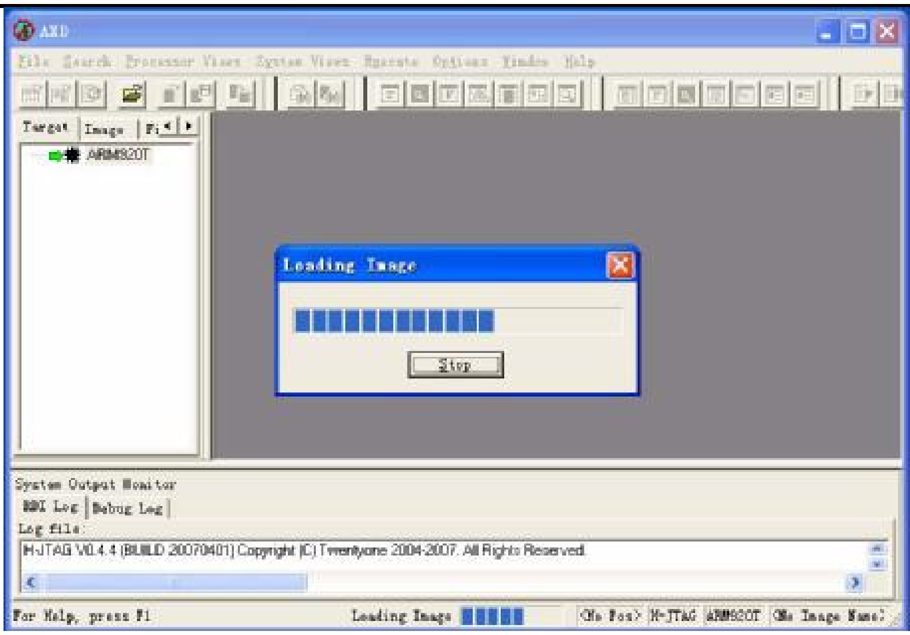


把光盘中“非操作系统示例代码”文件夹中的 2440test 目录复制到硬盘的某一个目录(在此为 D:\work)，去掉其只读属性，运行 ADS1.2 集成开发环境，点 File->Open…打开 2440test.mcp 文件，如图。

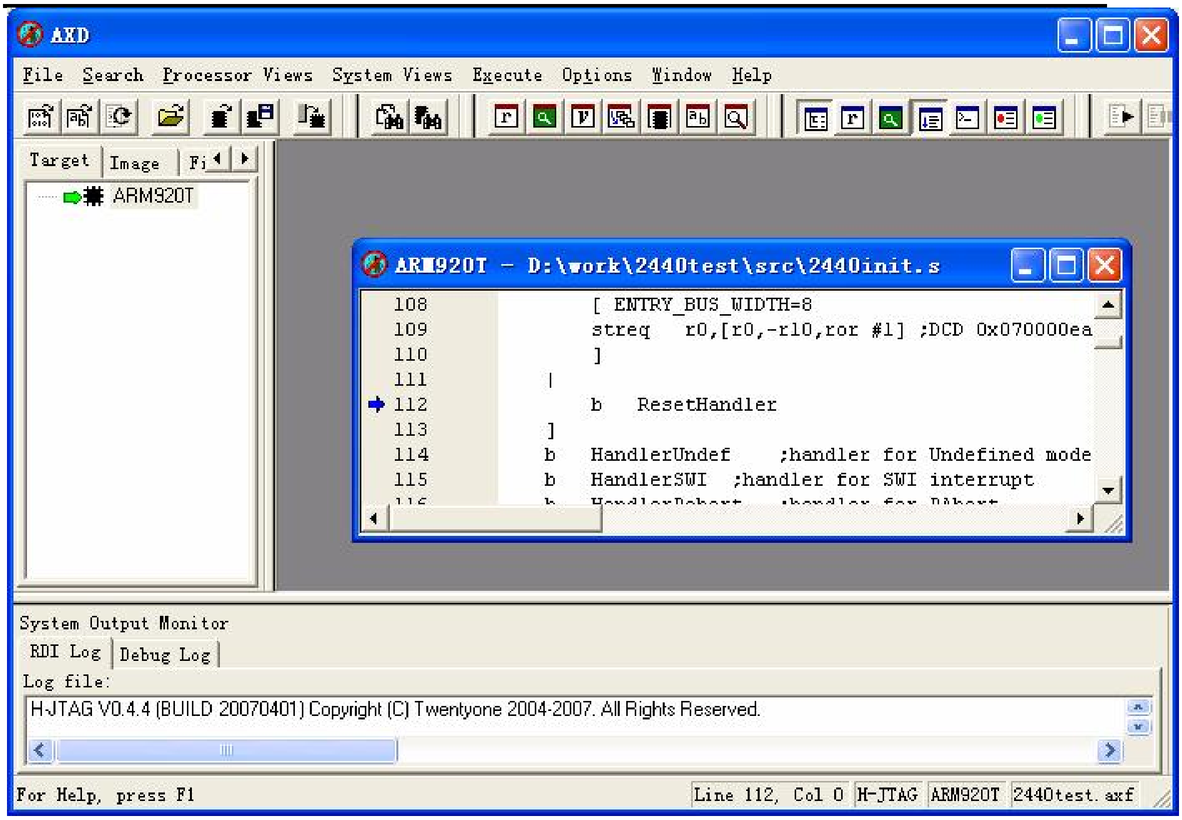


点 Project->Debug 或者按 F5 键开始编译 2440test 项目，编译完毕会自动启动 AXD Debugger 调试器，并把编译好的 2440test.axf 映象通过 JTAG 下载到内存中，如图。

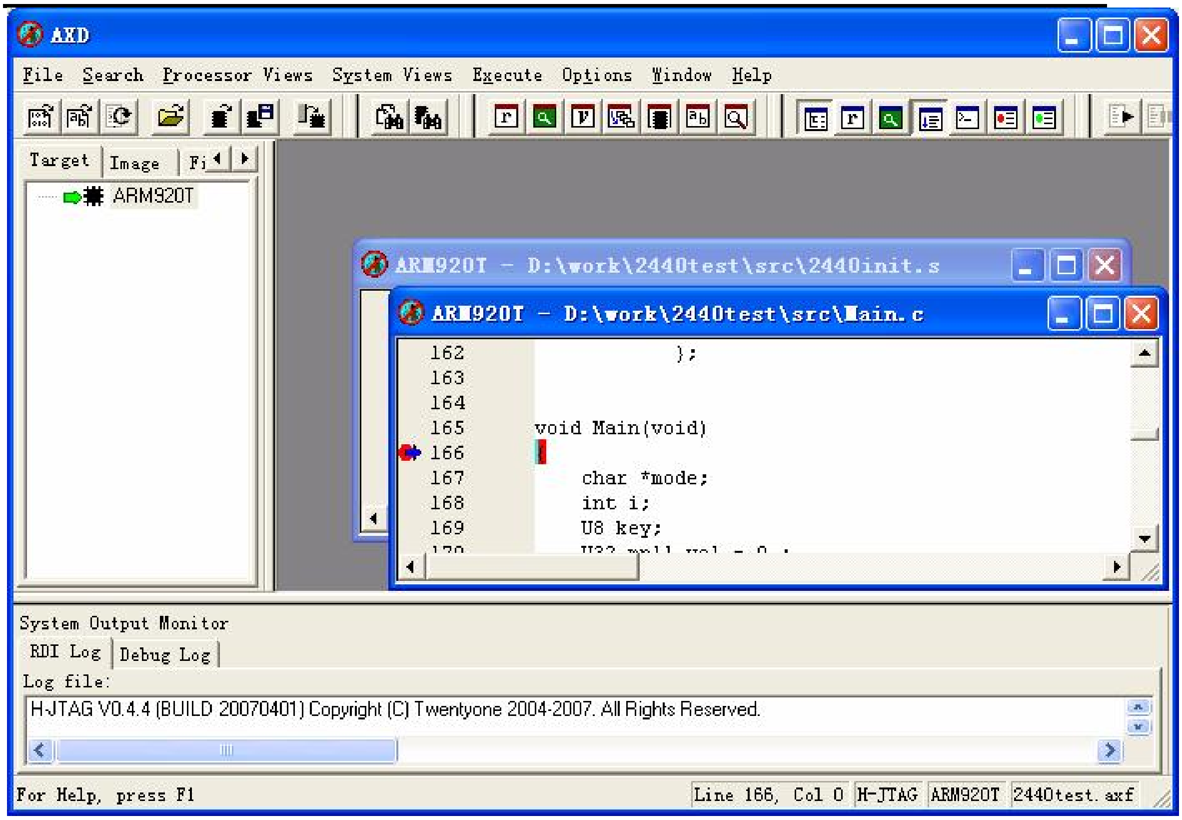
注意：因为 2440test.axf 比较大，因此下载需要等待一段时间。


AXD
File Search Processor Views System Views Execute Options Window Help
Target Image
ARM920T
Loading Image
Stop
System Output Monitor
RDI Log Debug Log
Log file:
H-JTAG V0.4.4 (BUILD 20070401) Copyright (C) Twentyone 2004-2007. All Rights Reserved.
For Help, press F1
Loading Image
H-JTAG
ARM920T

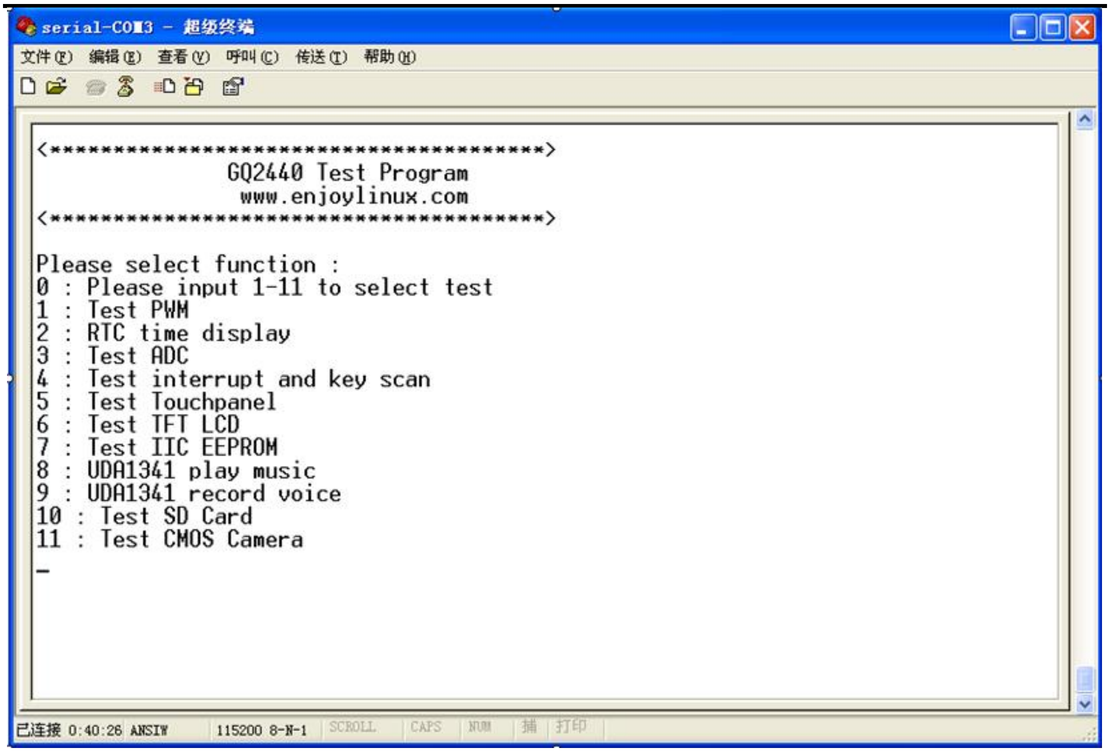
下载完毕如图所示。



这时点菜单 Execute->Go 或者按 F5 按键，程序将跳转到 Main 函数处，我们就可以进行单步运行调试了。



如果您这时再全速执行程序，可以看到从串口终端打印处如下信息，这和我们之前使用的非操作系统测试程序是一样的。



4.3.2 通过 USB 把 2440test 下载到内存运行

使用 USB 下载运行 2440test 程序不需要并口和 JTAG 板，借助 armvivi 的“Download & Run”功能就可以了，下面是详细的操作步骤：

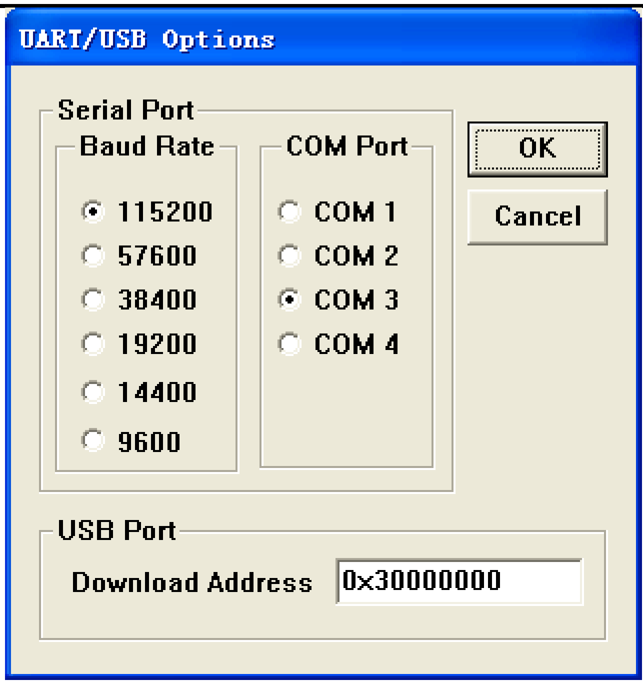
在光盘“images\2440test\”目录中找到 2440test.bin 文件，通过 BIOS 下载运行该测试程序，步骤如下：

(1)连接好开发板电源，串口线，USB 线，并通过按键开关 Nor_KEY 从 Nor Flash 启动系统，分别打开串口超级终端和 DNW，上电启动开发板。

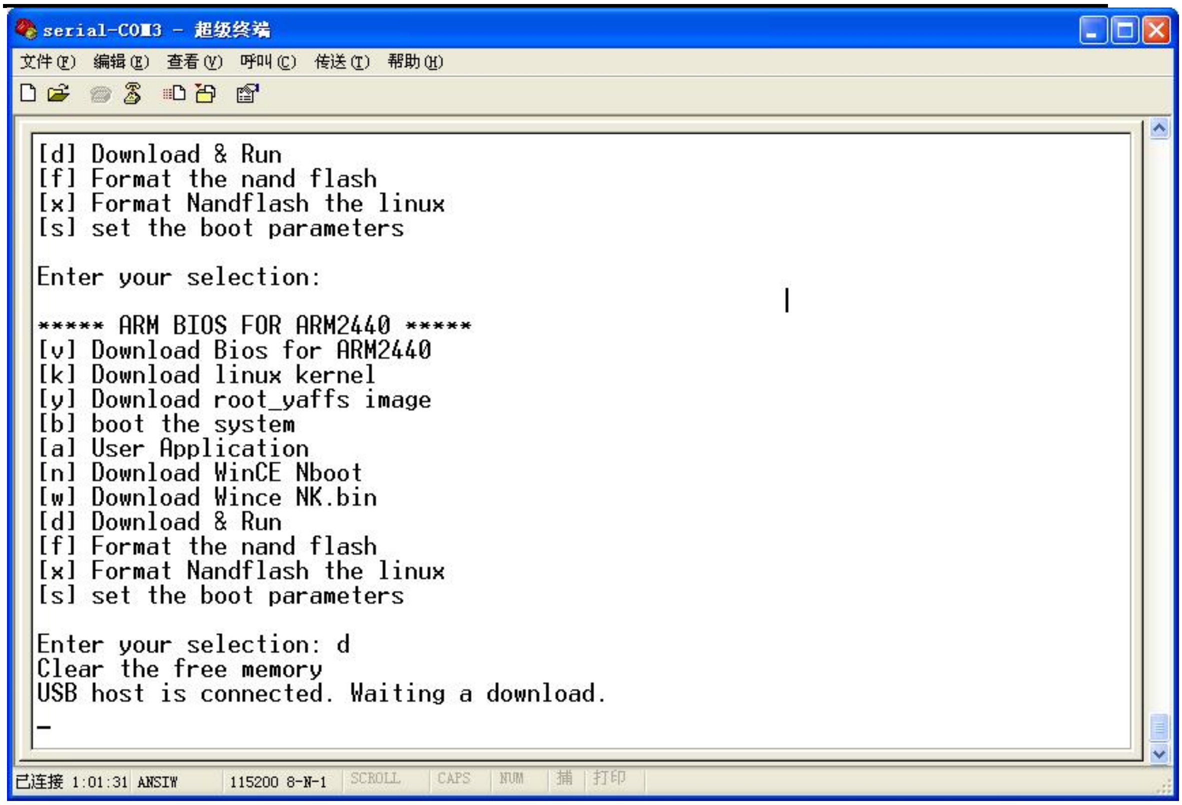
(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：



(3).点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000



(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息:



serial-COM3 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
[d] Download & Run
[f] Format the nand flash
[x] Format Nandflash the linux
[s] set the boot parameters

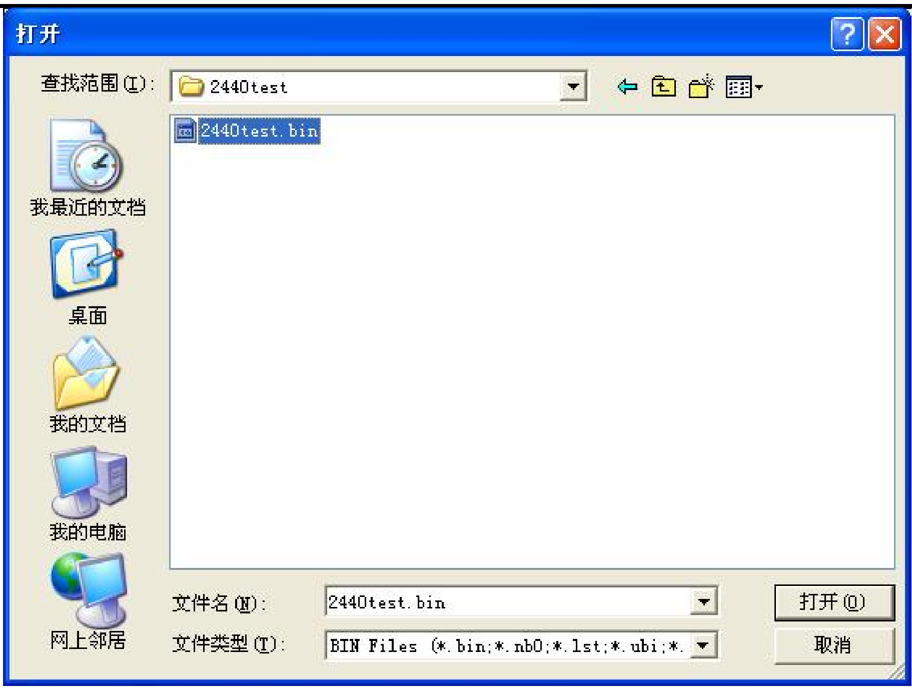
Enter your selection:

***** ARM BIOS FOR ARM2440 *****
[v] Download Bios for ARM2440
[k] Download linux kernel
[y] Download root_yaffs image
[b] boot the system
[a] User Application
[n] Download WinCE Nboot
[w] Download Wince NK.bin
[d] Download & Run
[f] Format the nand flash
[x] Format Nandflash the linux
[s] set the boot parameters

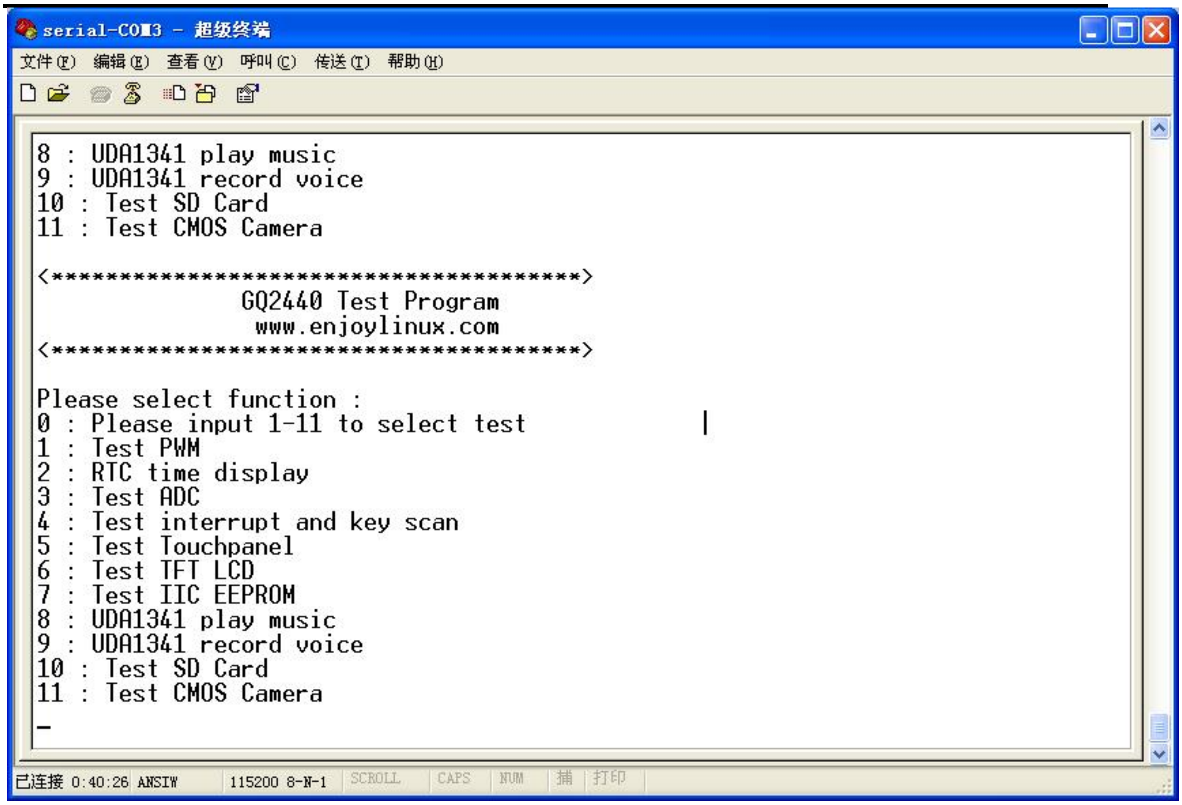
Enter your selection: d
Clear the free memory
USB host is connected. Waiting a download.
_
```

已连接 1:01:31 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

(5).点击 DNW 程序的"USB Port" "Transmit"，选择 2440test.bin 这个映象文件(在光盘的 images\2440test\目录下面)，接着点"打开"，这样就开始下载了。



(6)下载结束后，会自动运行，终端出现如下界面。



4.3.3 把 2440test 烧写到 Nand Flash 运行

说明：2440test 可以自动适应 64M/128M Nand Flash GQ2440，对于同一种 LCD 型号套餐，烧写的文件是一样的。

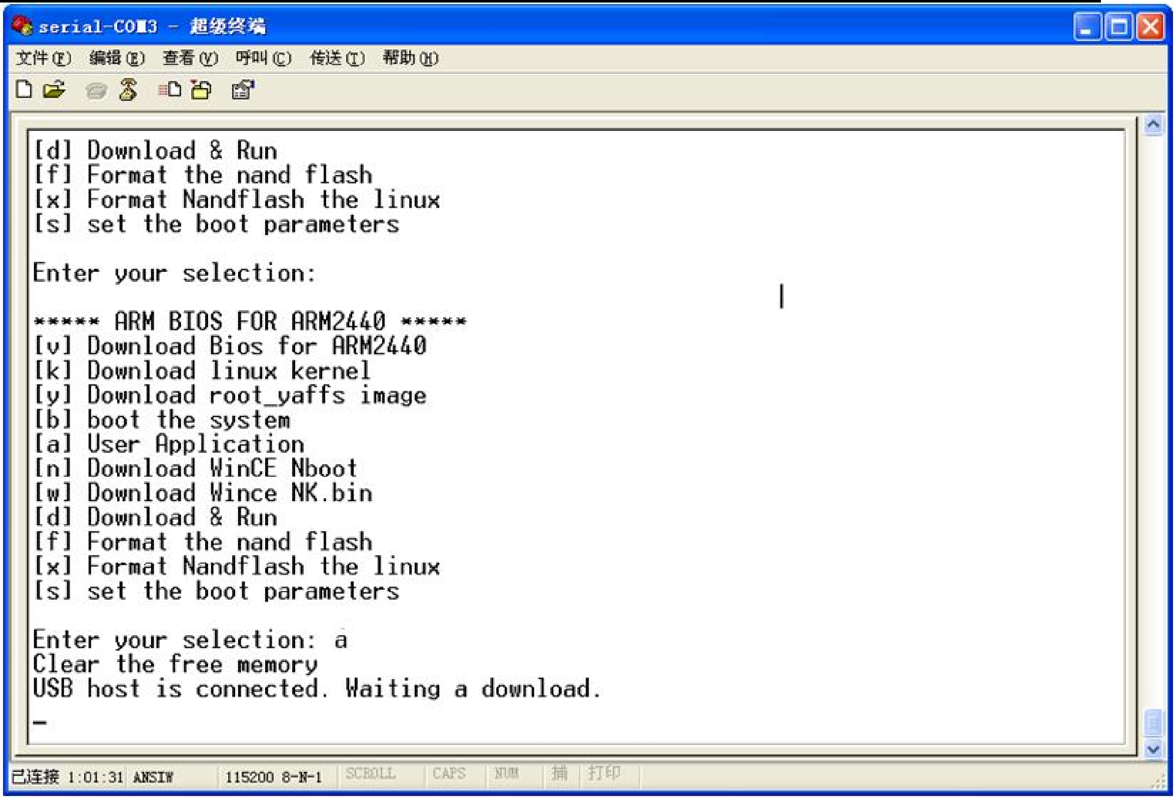
使用 armvivi 的功能号[a]可以把 2440test.bin 可执行程序烧写到 Nand Flash 中运行，

(1)连接好开发板电源，串口线，USB 线，并通过按键开关 Nor_KEY 从 Nor Flash 启动系统，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB: OK]，如果没有安装好驱动会显示[USB: x]，如图所示：



(4)这时在超级终端的 BIOS 功能菜单中选择功能号[a]，出现 USB 下载等待提示信



(5).点击 DNW 程序的【USB Port】|【Transmit】，如图选择刚刚编译出的映象文件(光盘“\images\2440test\”目录中有已经编译好的可执行文件)，这样就开始下载了，下载完毕，armvivi 会把它自动烧写到 Nand Flash 起始块为 0 的地方，即 Block 0 开始处。

烧写完毕，按下 POW_KEY 使开发板从 Nand Flash 启动就可以了。

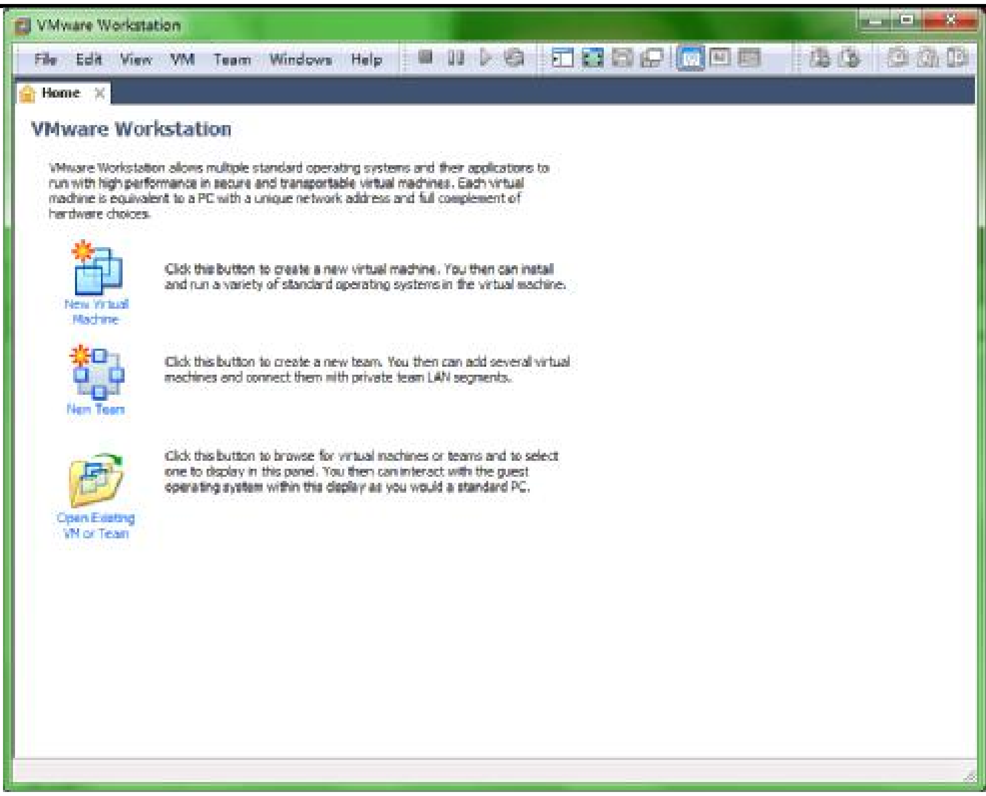
第5章 建立 Linux 开发环境

注意： 自从 Linux-2.6.29+Qtopia-2.2.0 开始(本开发板所配最新内核为 Linux-2.6.32.2)，
我们就基于 RHEL5(即红帽企业版 5) 平台做开发，所有的配置和编译脚本也基于此平台，我们没有在其他平台上测试过。如果你对 Linux 开发很熟悉，相信你会根据错误提示逐步找到原因并解决，它们一般是你选用的平台缺少了某些库文件或者工具等原因造成的；否则，我们建议初学者使用和我们一致的平台，即 RHEL5(全称为：RHEL5.1-Server-20071017.0 -i386-DVD.iso)，你可以在我们网站下载(http://www.enjoylinux.cn/ ，不保证长期有效)，也可以在其他地方获取，它们都是一样的，安装时请务必参考我们手册提供的步骤，这是我们经过严格测试的，以免遗漏一些开发时所需要的组件。

Linux 的发行版本众多，我们无法为此一一编写文档解释安装方法，请谅解。

5.1 图解安装 RHEL5

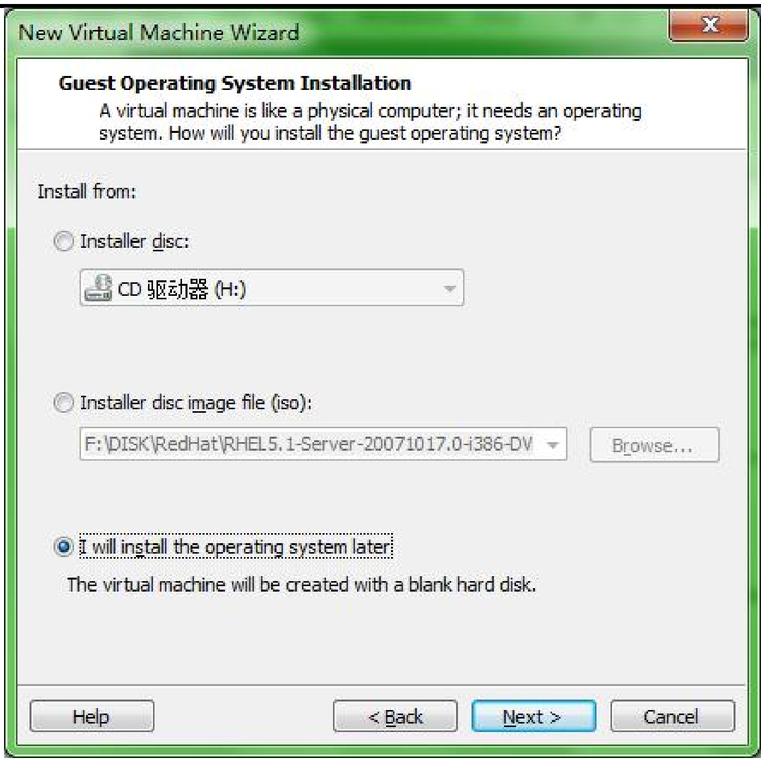
1. 在 windows 中安装好虚拟机虚拟机（安装文件位于开发板光盘目录**\windows 平台工具\Vmware**）后，运行该程序，会出现如图所示的界面。



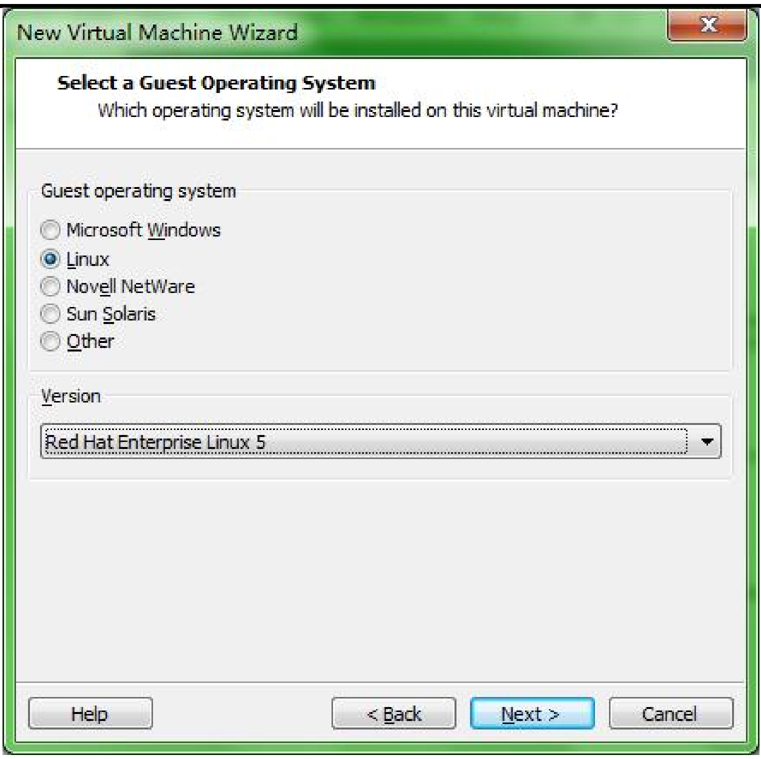
2. 通过【File】|【New】|【Virtual Machine】，或者点击图标(New Virtual Machine)，打开新建虚拟机向导，如图所示，选择 Custom (advanced) 表示用户定制。



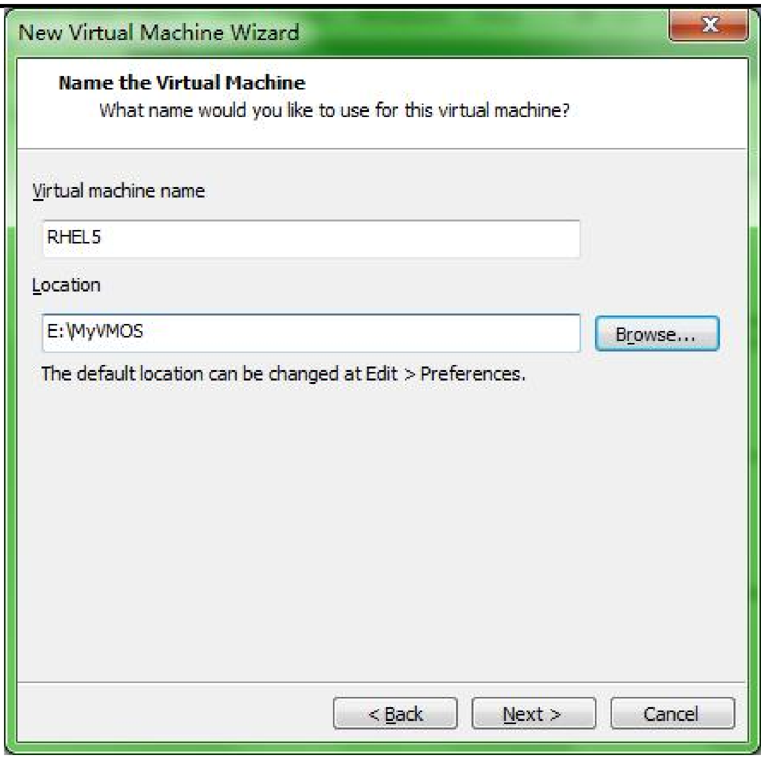
3. 点击 Next > 选择下一步，在出现的界面中继续选择下一步，直到出现如图所示的界面。选择 I will install the operating system later ，表示先设置好虚拟机环境之后再安装操作系统。



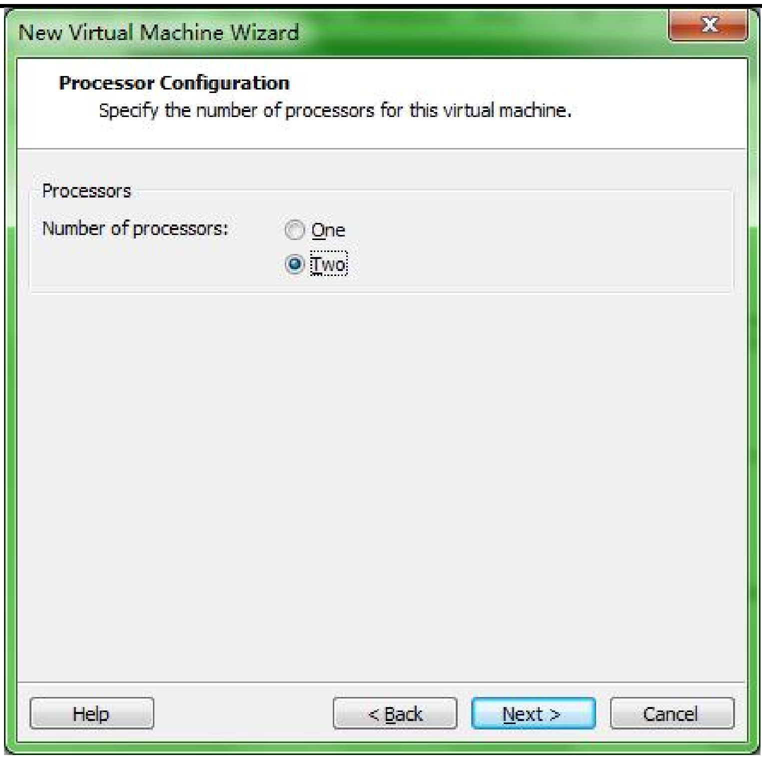
4. 在接下来的界面中选择 Linux，并在 Version 选项中选择 Red Hat Enterprise Linux 5，单击 Next 进入下一步，如图所示。



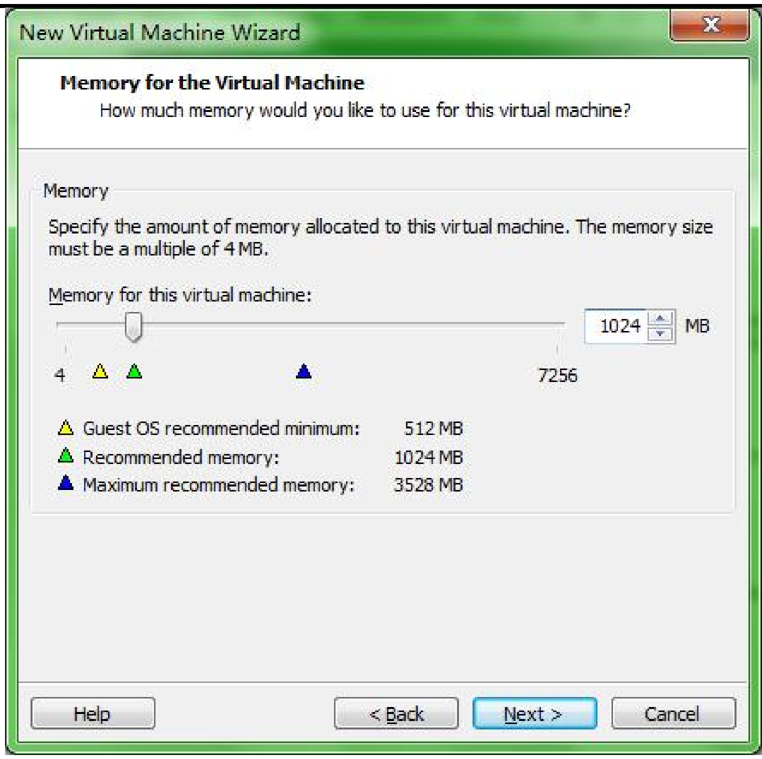
5. 在出现的界面中，选择要安装 RHEL5 的位置，如图所示。并单击 Next 进入下一步。



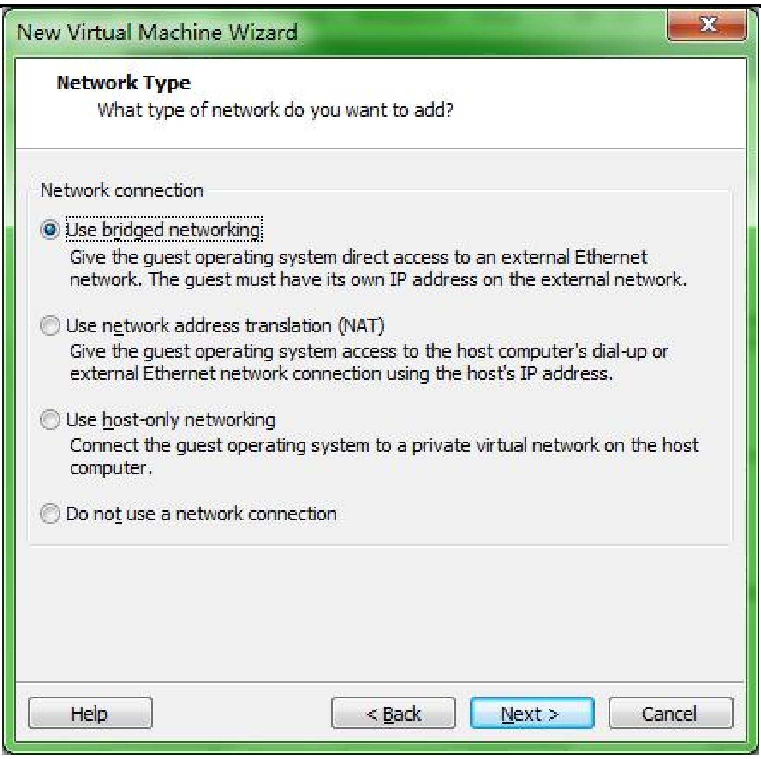
6. 选择 cpu 是否是双核，如果是双核选择 Two，如图所示，单击 Next 进入下一步。



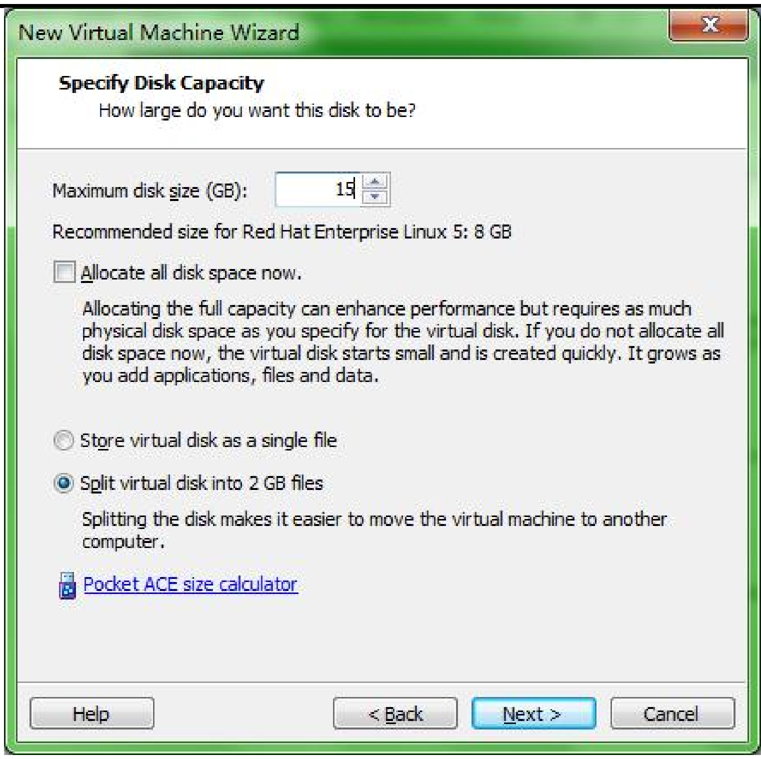
7. 选择给虚拟机分配的内存大小，pc 的内存较大的话可以选择 1024MB，根据自己 pc 物理内存大小来处理。但是如果要运行图形界面的话建议在 256MB 以上。如图所示。



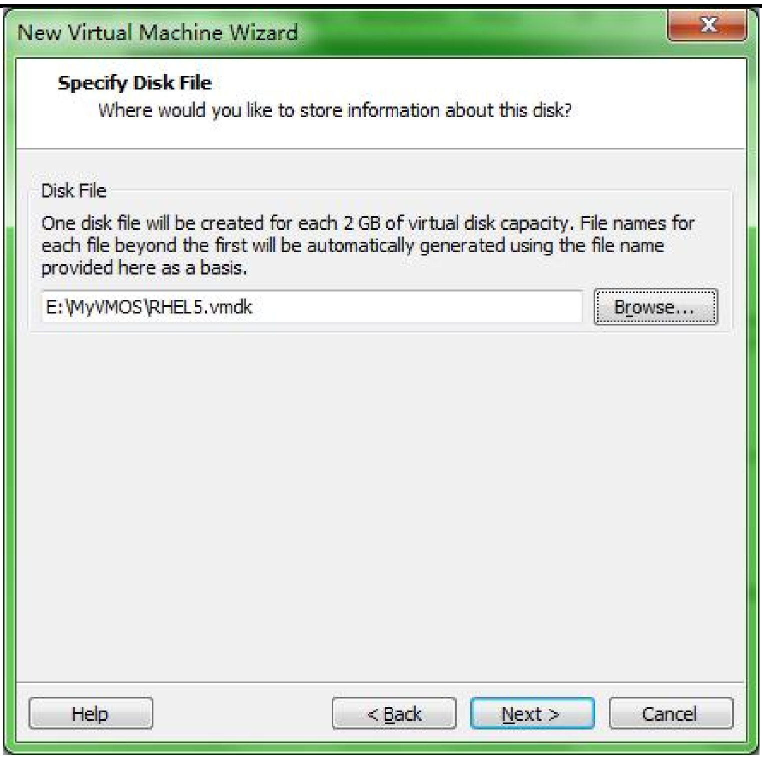
8. 单击 Next 进入下一步，选择网络连接方式，选择 Use bridged networking ，即桥接方式，如图所示。



9. 单击 Next 进入下一步，直到出现如图所示的画面，磁盘大小可以根据需要选择分配，一般推荐分配为 15G 以上，虚拟磁盘划分选择 Split virtual disk into 2 GB files 。

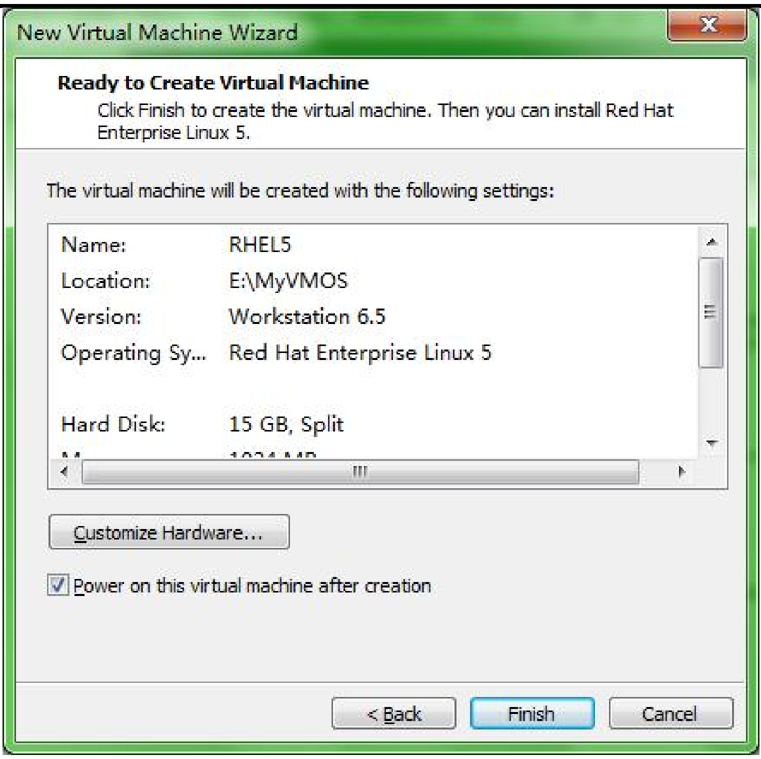


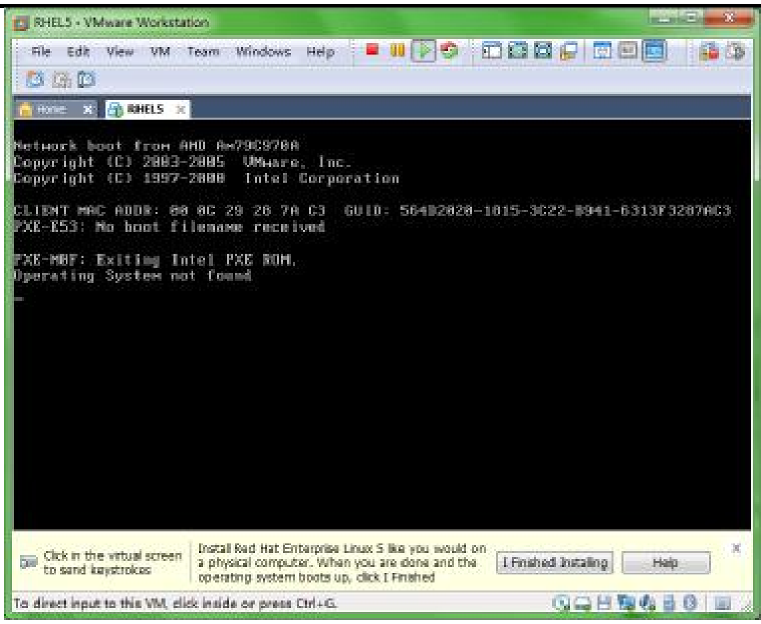
10. 单击 Next 进入下一步，选择虚拟机磁盘文件 RHEL.vmdk 存放的位置，推荐设置为之前设置的虚拟机安装目录。如图所示。



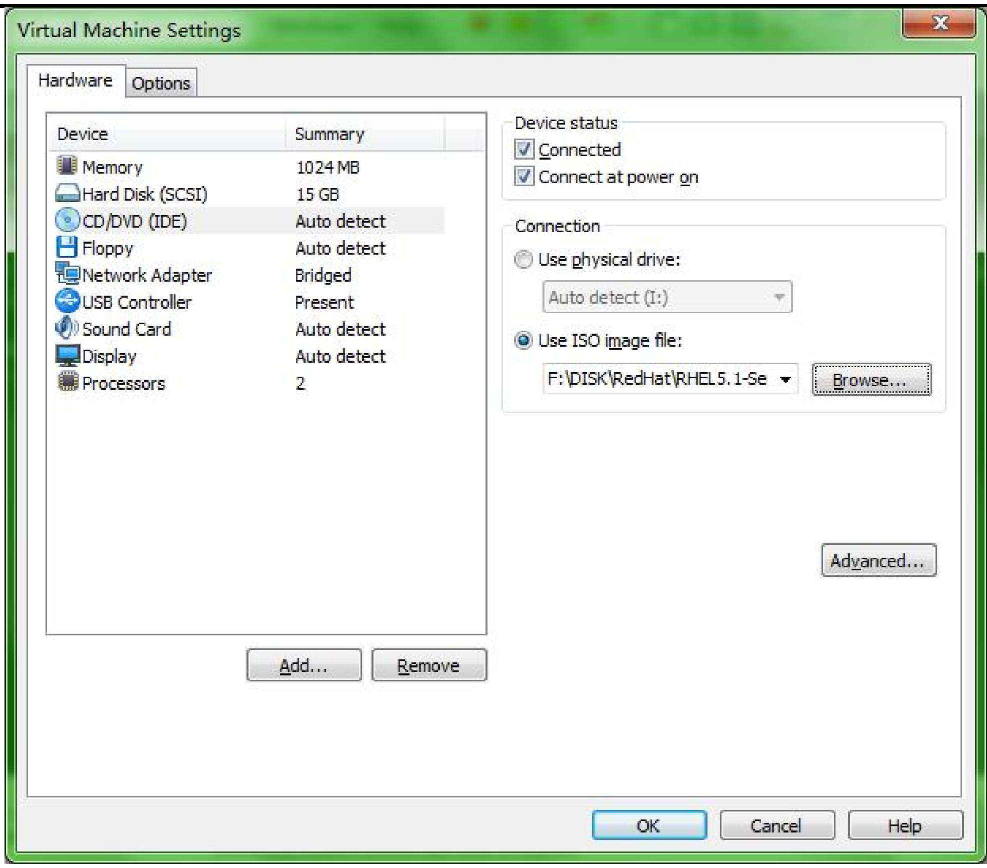
11. 单击 Next 进入下一步，单击 

完成安装 RHEL5 的虚拟机设置。这时，界面会进入虚拟机的启动状态，会显示找不到操作系统，如图所示。

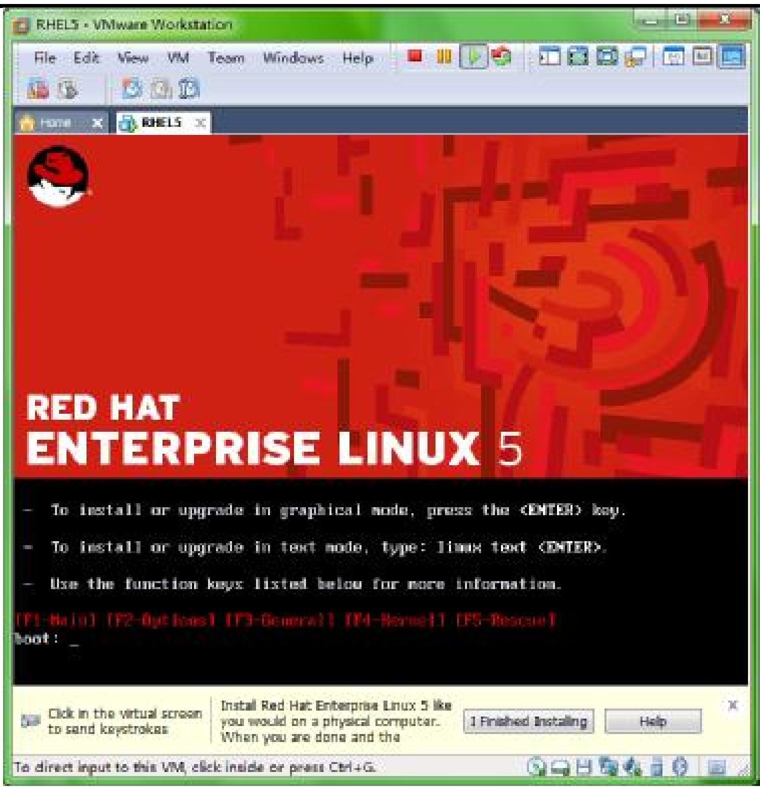

New Virtual Machine Wizard
Ready to Create Virtual Machine
Click Finish to create the virtual machine. Then you can install Red Hat Enterprise Linux 5.
The virtual machine will be created with the following settings:
Name: RHEL5
Location: E:\MyVMOS
Version: Workstation 6.5
Operating Sy... Red Hat Enterprise Linux 5
Hard Disk: 15 GB, Split
Customize Hardware...
Power on this virtual machine after creation
< Back
Finish
Cancel



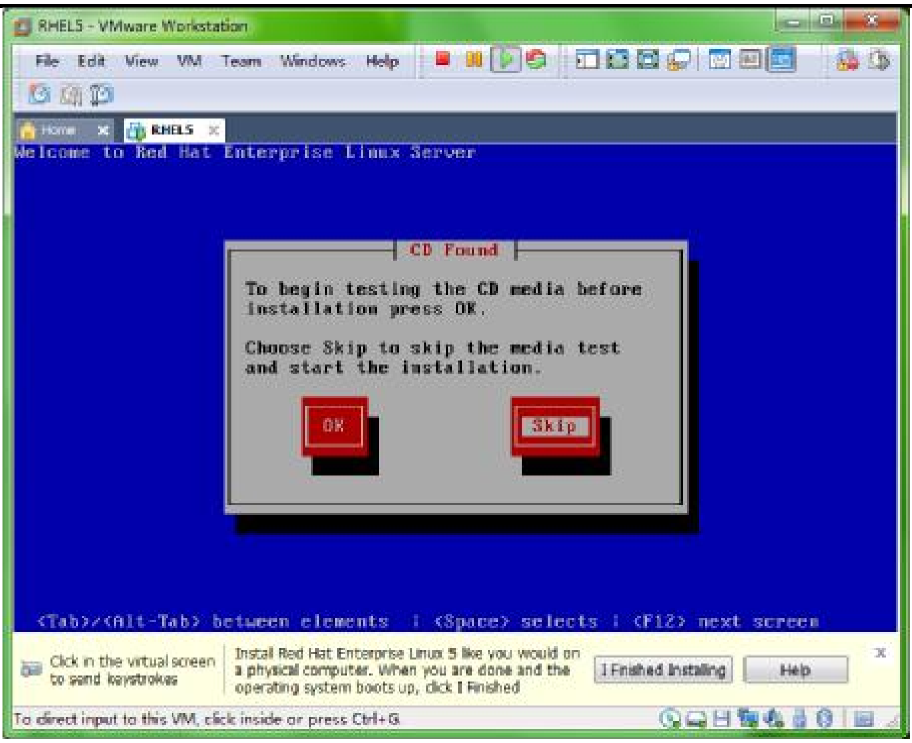
12. 用虚拟机的光驱加载 RHEL5 的安装镜像文件。通过下拉菜单【VM】|【Settings】，打开如图的界面。在 Device 列表框中选择 CD/DVD (IDE)，并在 Connect 项中选择 Use ISO image file: ，并通过 Browse... 浏览并选择要安装的 RHEL5 镜像文件。点击 OK 确认退出。



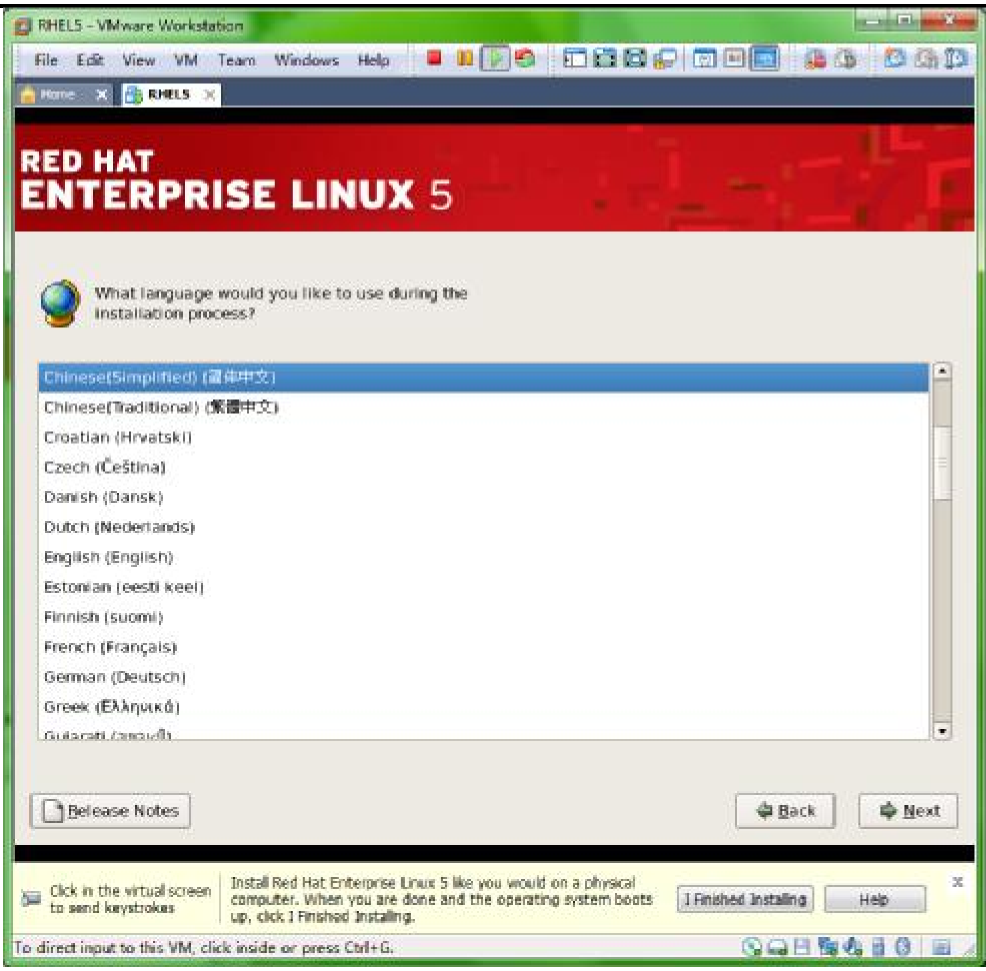
13. 单击虚拟机主界面工具栏上的图标重启虚拟机，会出现如图的画面。

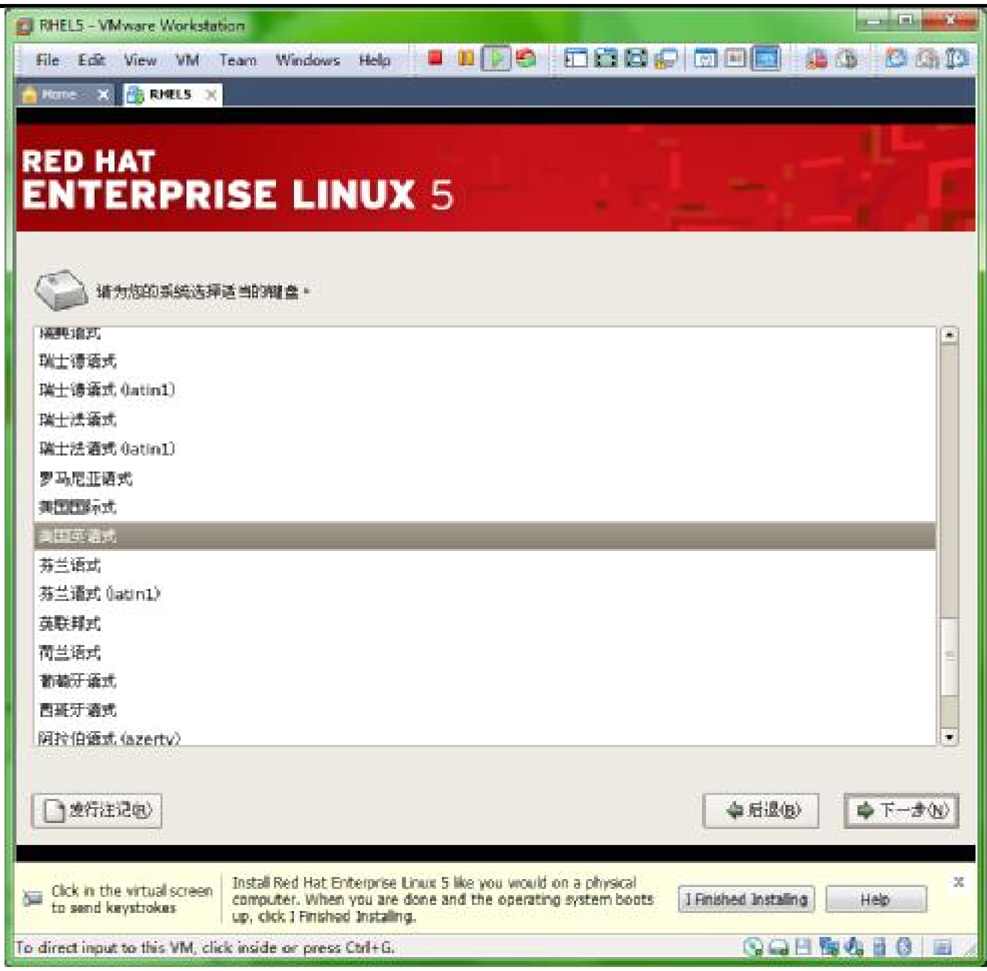


14. 选择是否检测光盘的完整性，选择 Skip 跳过，如图所示，然后正式进入了 RHEL5 的安装向导界面。点击 Next 进入下一步，选择操作系统语言，这里选择了简体中文，推荐使用默认的语言。点击 Next 进入下一步，选择键盘语言，使用默认的"美国英语式"。如图所示。


RHEL5 - VMware Workstation
File Edit View VM Team Windows Help
Home RHEL5
Welcome to Red Hat Enterprise Linux Server
CD Found
To begin testing the CD media before installation press OK.
Choose Skip to skip the media test and start the installation.
OK
Skip
<Tab>/<Alt-Tab> between elements | <Space> selects | <F12> next screen
Click in the virtual screen to send keystrokes
Install Red Hat Enterprise Linux 5 like you would on a physical computer. When you are done and the operating system boots up, click I Finished
I Finished Installing
Help
To direct input to this VM, click inside or press Ctrl+G.

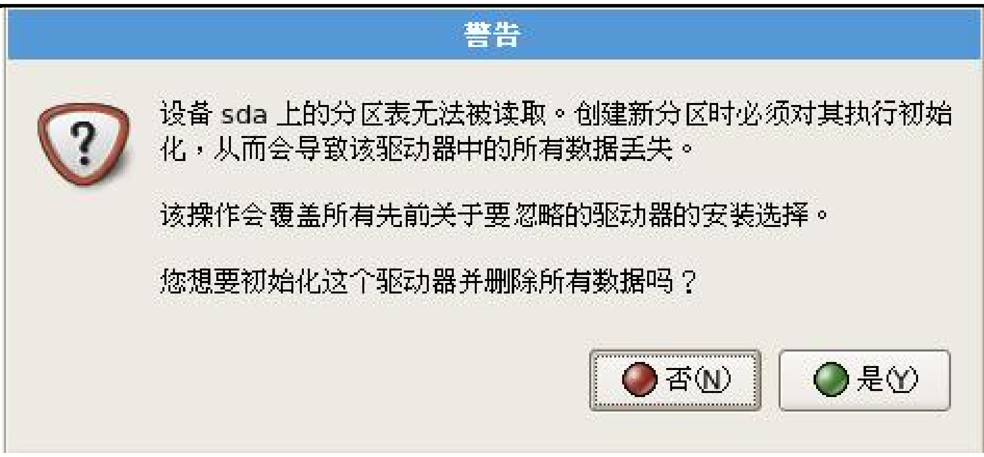

RHEL5 - VMware Workstation
File Edit View VM Team Windows Help
RED HAT
ENTERPRISE LINUX 5
RED HAT
ENTERPRISE LINUX 5
Release Notes
Back
Next
To direct input to this VM, click inside or press Ctrl+G


RHEL5 - VMware Workstation
File Edit View VM Team Windows Help
Home
RHEL5
RED HAT
ENTERPRISE LINUX 5
What language would you like to use during the installation process?
Chinese(Simplified) (简体中文)
Chinese(Traditional) (繁體中文)
Croatian (Hrvatski)
Czech (Čeština)
Danish (Dansk)
Dutch (Nederlands)
English (English)
Estonian (eesti keel)
Finnish (suomi)
French (Français)
German (Deutsch)
Greek (Ελληνικά)
Release Notes
Back
Next
Click in the virtual screen to send keystrokes
Install Red Hat Enterprise Linux 5 like you would on a physical computer. When you are done and the operating system boots up, click I Finished Installing.
I Finished Installing
Help
To direct input to this VM, click inside or press Ctrl+G.



15. 单击“下一步”，弹出“安装号码”的界面，选择 跳过输入安装号码，确定。之后又会出现询问是否初始化磁盘的对话框，选择 是(Y)，如图所示。


RHEL5 - VMware Workstation
File Edit View VM Team Windows Help
Home RHEL5
RED HAT
ENTERPRISE LINUX 5
请为您的系统选择适当的键盘。
瑞士德语式
瑞士德语式 (latin1)
瑞士法语式
瑞士法语式 (latin1)
罗马尼亚语式
美国国际式
美国英语式
芬兰语式
芬兰语式 (latin1)
英联邦式
荷兰语式
葡萄牙语式
西班牙语式
阿拉伯语式 (azerty)
安装号码
安装号码:
跳过输入安装号码
取消(C)
确定(O)
发行注记(R)
后退(B)
下一步(N)
To direct input to this VM, click inside or press Ctrl+G.



16. 磁盘初始化后会出现如下界面，这里使用默认的硬盘分区方案，点击“下一步”，出现确认对话框，选择，如图所示。


RHEL5 - VMware Workstation
File Edit View VM Team Windows Help
Home
RHEL5
RED HAT
ENTERPRISE LINUX 5
安装需要对您的硬盘进行分区。默认情况下，选择适合大多数用户的
分区方案。您可以选择默认方式，也可以创建您自己的分区方式。
在选定驱动上删除 linux 分区并创建默认的分区结构。
选择本次安装中使用的驱动程序(S)。
sda 15359 MB VMware, VMware Virtual S
(A) 高级存储配置
检验和修改分区方案(V)
发行注记(R)
后退(B)
下一步(N)
To direct input to this VM, click inside or press Ctrl+G.



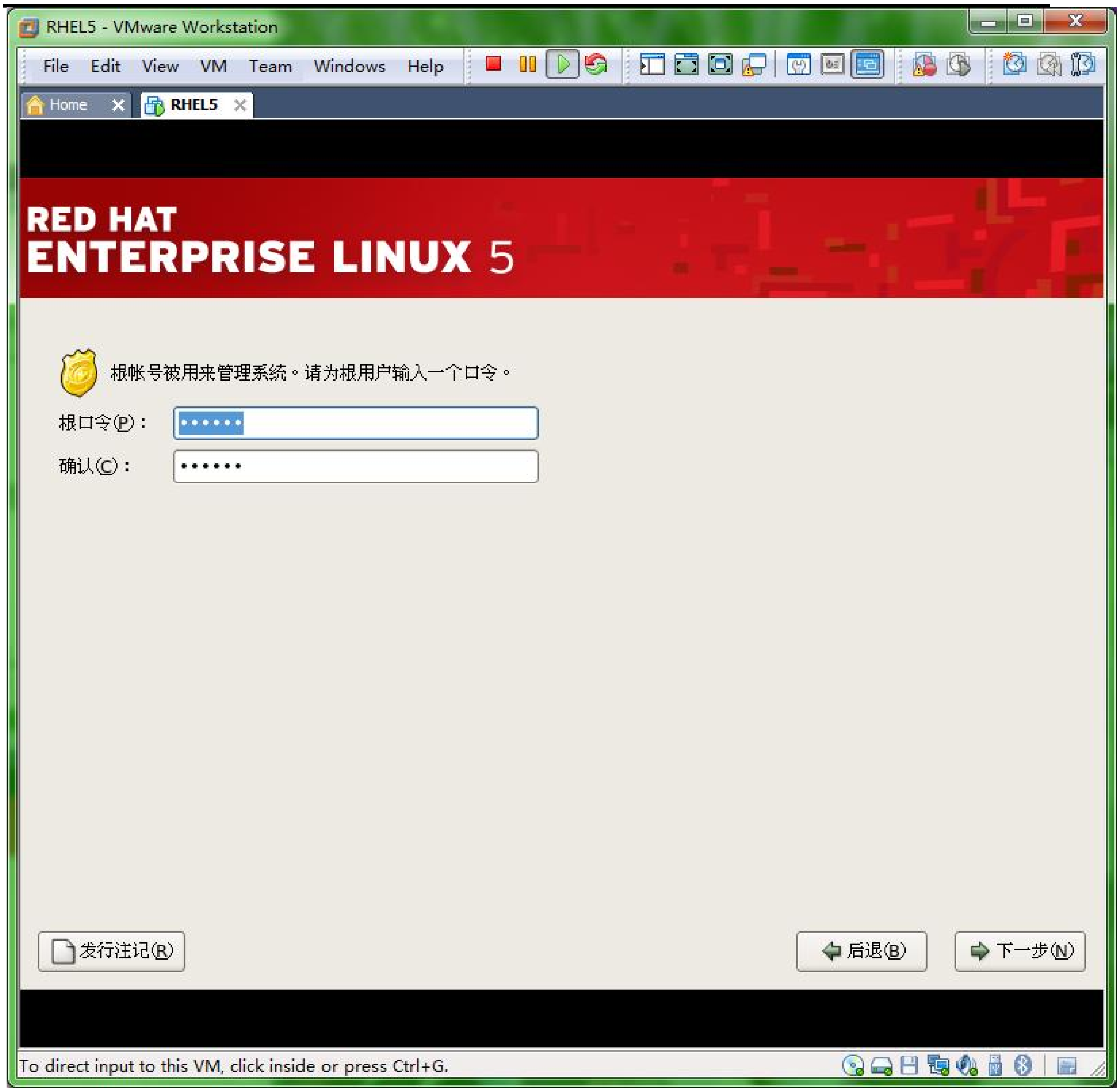
17. 这时进入了网络配置界面，使用默认的通过 DHCP 自动分配，选择“下一步”确认，如图所示。



18. 单击“下一步”，进入时区选择，可以在地图上选择合适的城市，如图所示。



19. 单击“下一步”，输入 root 账户的密码并确认，如图所示。



20. 单击“下一步”，进入到软件定制安装界面，如图所示，将“软件开发”和“网络服务器”的额外功能都选上，并且选择“现在定制”。



21. 单击“下一步”，进入到软件包选择界面。“桌面环境”用默认的“GNOME 桌面环境”即可，“应用程序”也用默认的，“开发”要把“老的软件开发”选上。主要是“服务器”的选择，最好全部选上，“基本系统”也可以全部选上，“语言支持”用默认的即可。

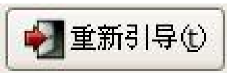


注意：上面的软件选择方案是为了尽量节省虚拟机的磁盘空间，如果装上系统后发现还有没安装全的软件包，可以直接在 linux 中加载 RHEL5 光盘镜像安装。如果你的磁盘

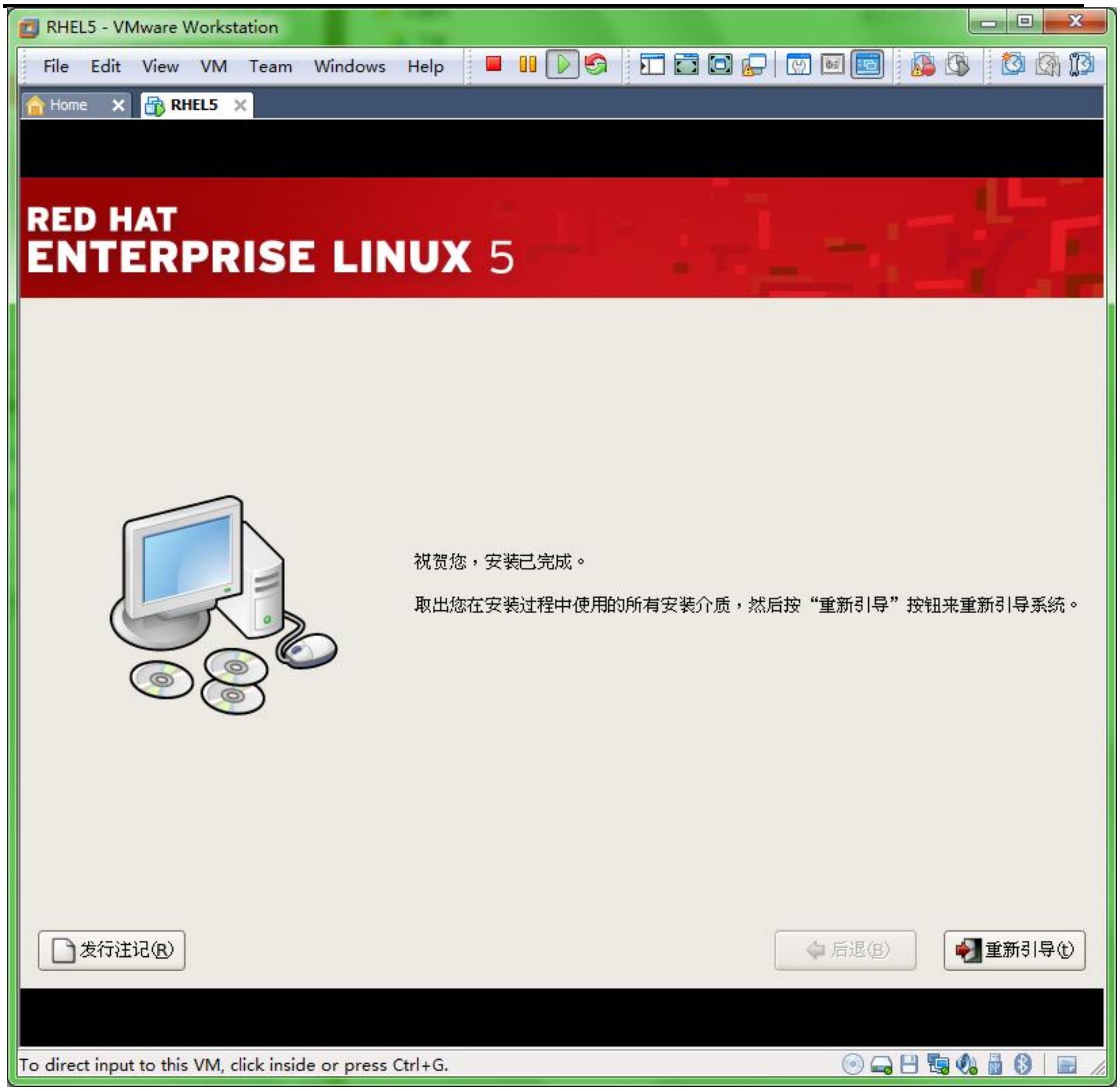
够大(大于 15G)，你完全可以全部安装这些软件。要想装全 RHEL5 提供的软件，除了选择右边列表框中的所有项目外，还要通过**，进入到明细选择界面进行选择。**

22. 单击“下一步”，会开始检测软件安装包的依赖关系，检测完毕后会进入到确认配置界面。再次单击“下一步”，就开始进行 RHEL5 的安装了，如图所示，安装过程需要一定时间。



23. 安装完成后出现如图所示画面，点击

。



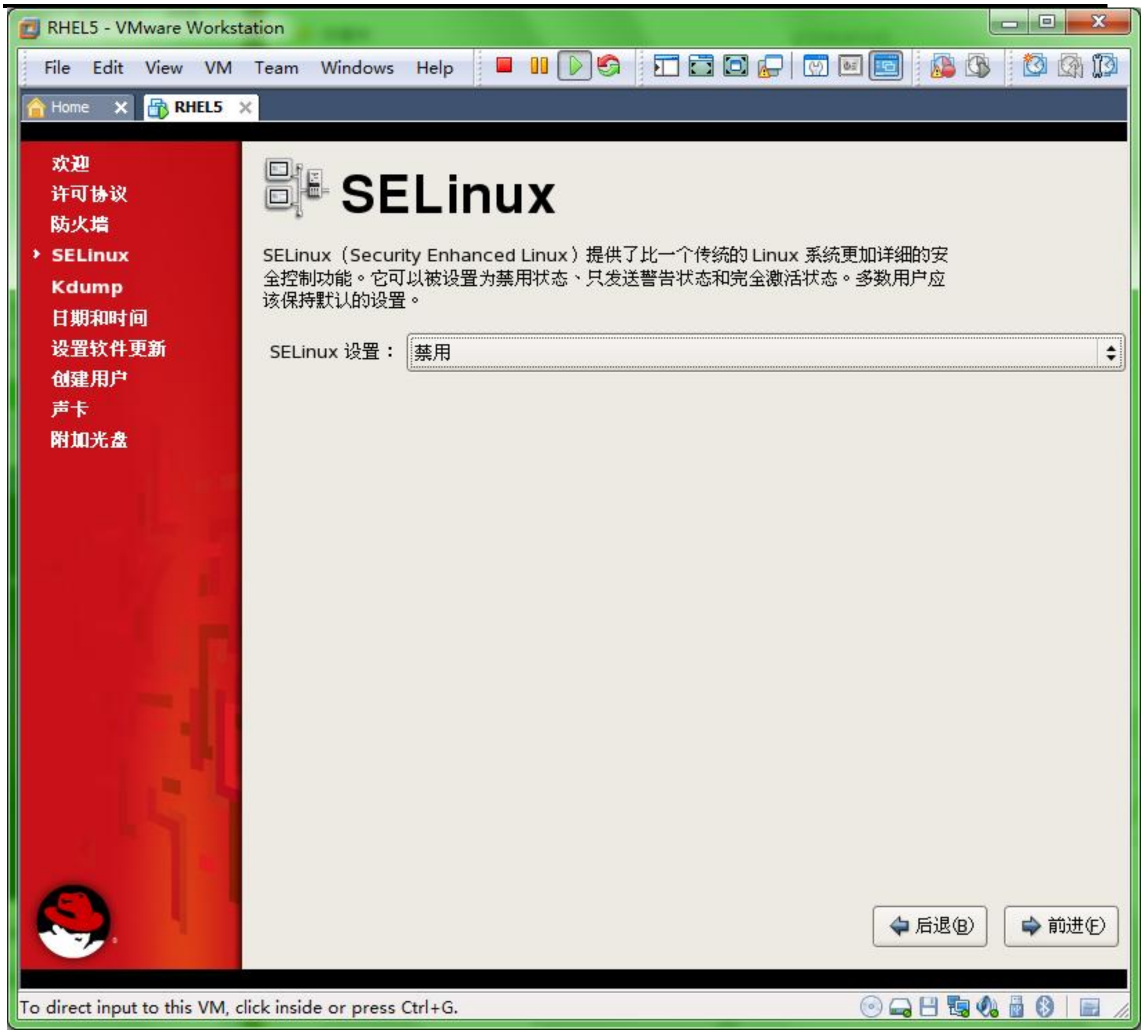
24. 重新引导后，系统进入后续配置阶段，如图所示。



25. 单击“前进”，进入到防火墙配置界面，如图所示。选择禁用防火墙，或将“信任的服务”全部选中。



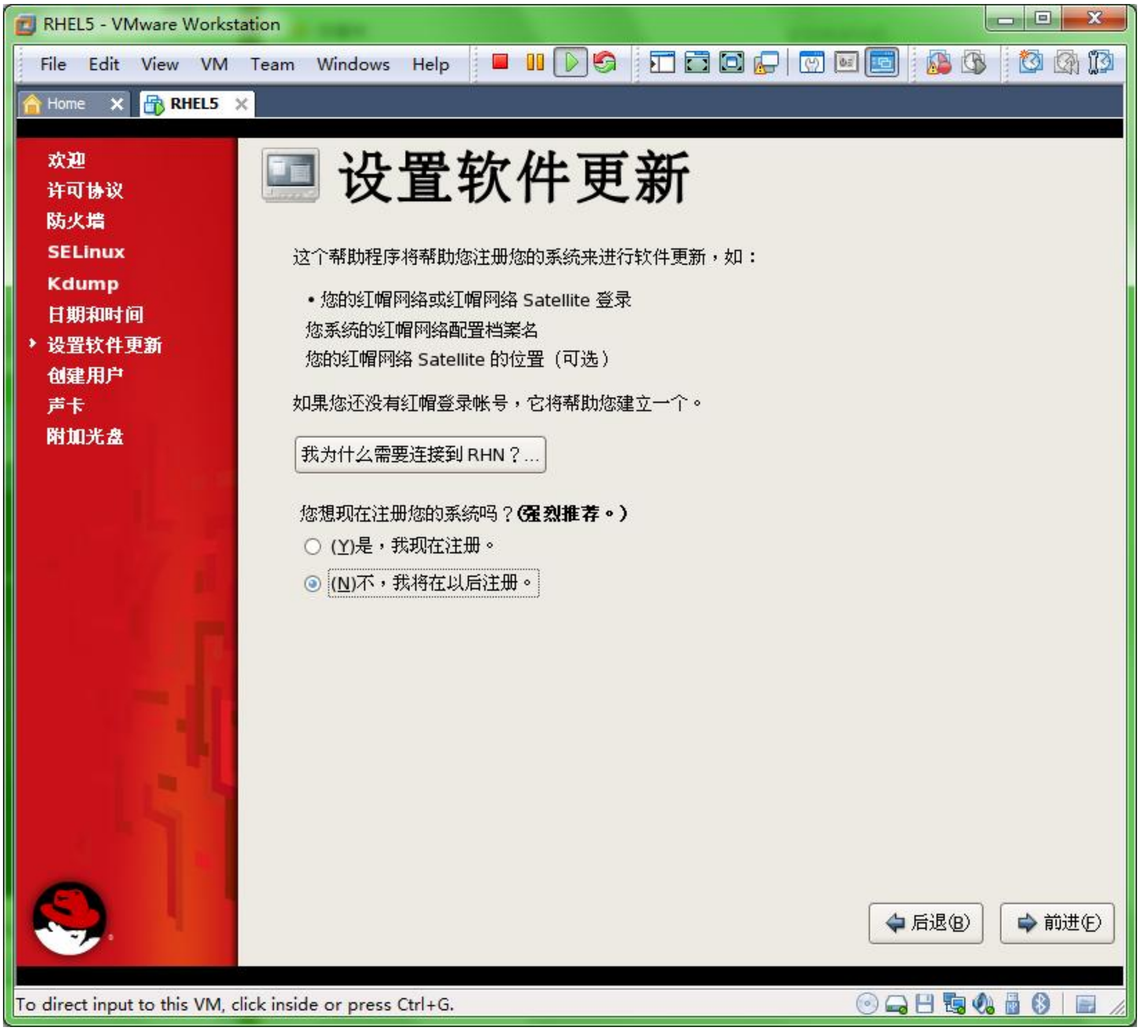
26. 单击“前进”，进入到 SELinux 配置界面，如图所示，选择禁用掉 SELinux。



注意：这里禁用防火墙和 SELinux 主要是因为它们会限制 Samba 服务。这样以后我们配置 Samba 服务器时就不需要在设置了。

27. 之后的两步分别是“Kdump”和“日期和时间”，“Kdump”可以不启用，也可以启用后使用默认配置，“日期和时间”根据本地时间调整即可。

28. 单击“前进”，进入“设置软件更新”界面，选择 (N)不，我将在以后注册。，如图所示。



29. 单击“前进”，进入“创建用户”界面，创建一个普通用户，如图所示。



30. 接下来的两步是"声卡"和"附加光盘","声卡"可点击 进行测试，"附加光盘"不需要安装，直接点击 。这时，整个 RHEL5 就安装成功了，系统重启后会看到如图所示的登录界面。可以输入前面安装过程中设置了密码的 root 账户或者创建的普通账户登录。



5.2 常用设置和服务

现在介绍一些开发中常用的设置和服务，其实还有很多 Linux 命令也是需要掌握的，用户可以自己找本 Linux 的书籍看看。

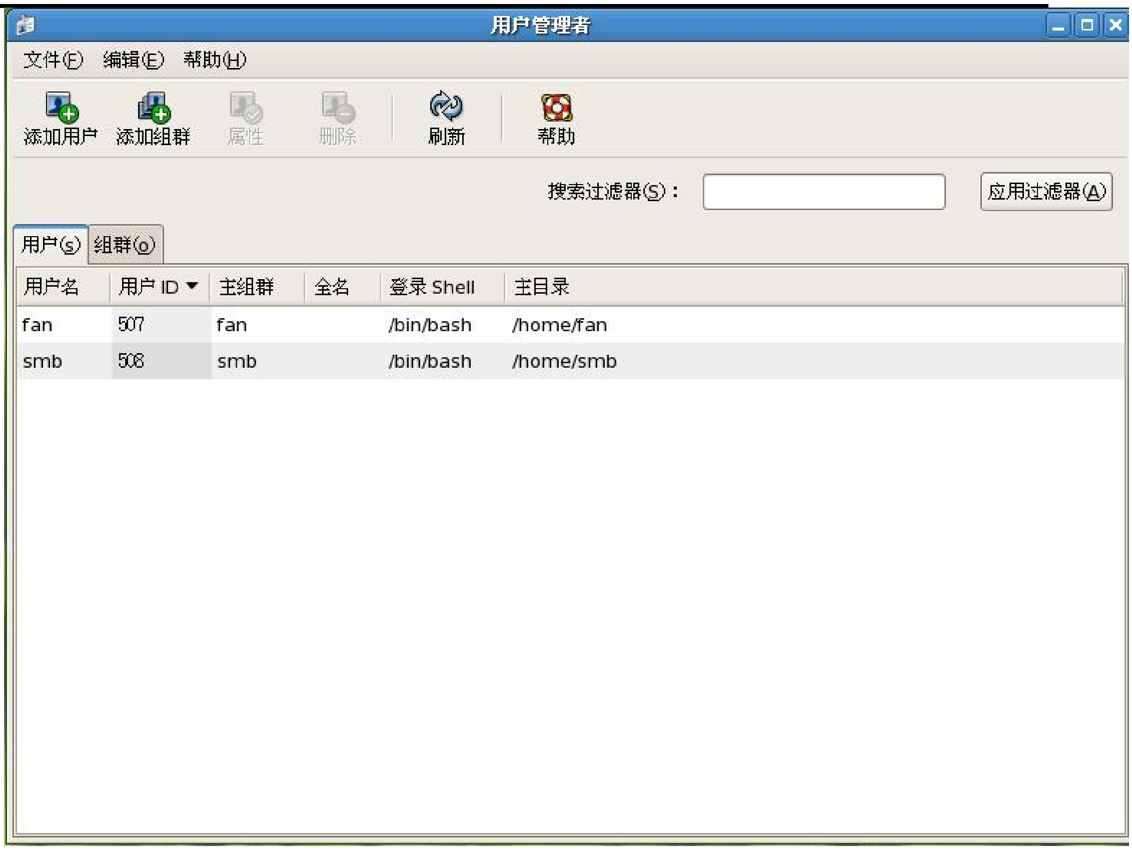
5.2.1 添加新用户

为了方便开发，我们通常创建一个普通权限的用户，步骤如下。

Step1：如图打开用户和组管理器：



Step2：出现用户管理窗口：



Step3：点工具栏的【添加用户】（英文系统为【Add User】)按钮，添加新用户，并设置密码:



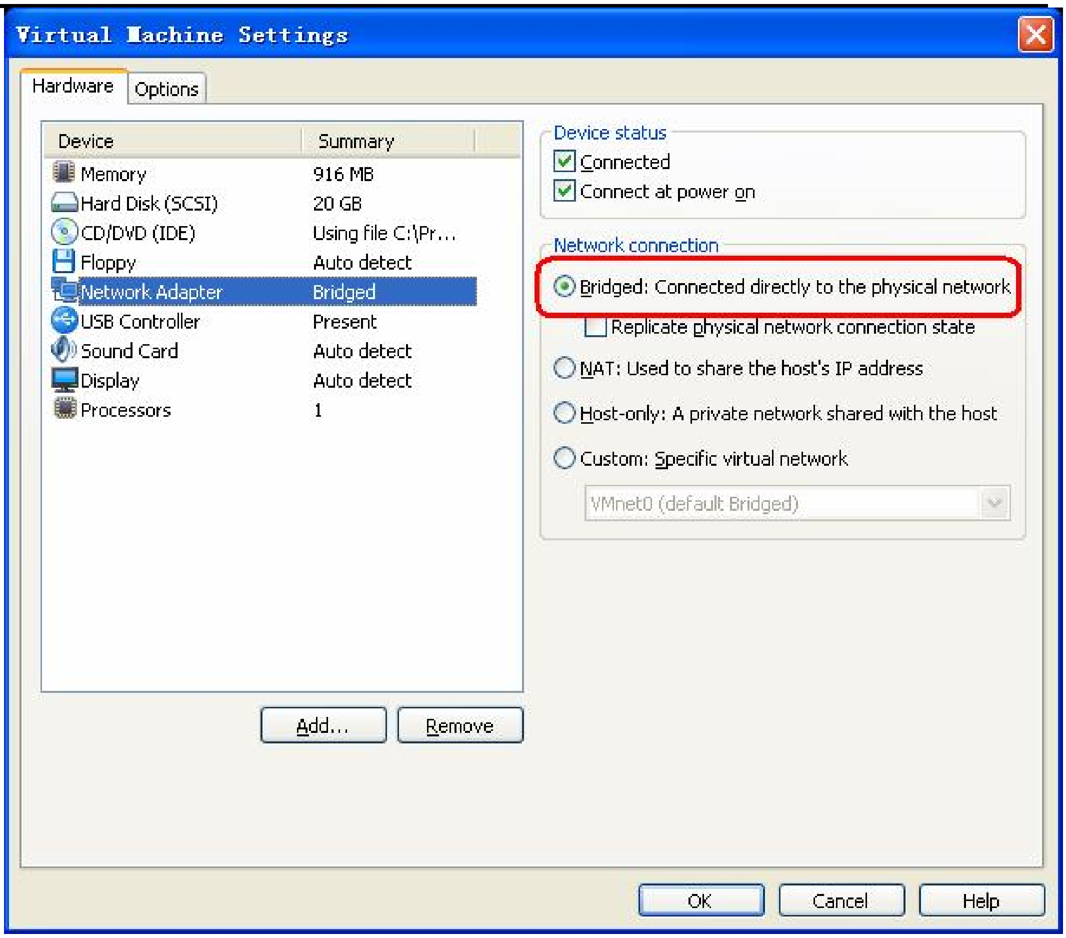
点【确定】返回，可以看到已经增加了 GQ 用户，同时/home 目录下也增加了 GQ 用户目录，如图：



5.2.2 访问 Windows 系统中的文件

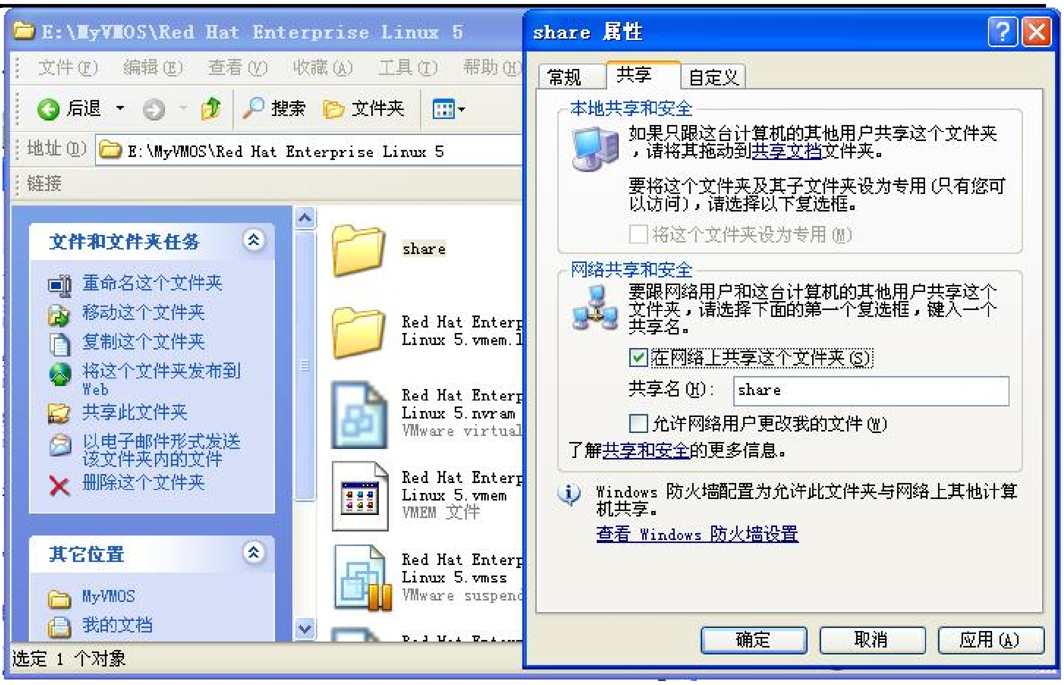
无论你使用的是虚拟机还是真实的 Fedora9 系统，都可以很方便的访问 Windows 中的共享文件，前提是两个系统之间的网络是互通的。

提示：要在虚拟机中使用网络，最简单的方式是设置"Guest"为"Bridges"方式的网络连接，如图：



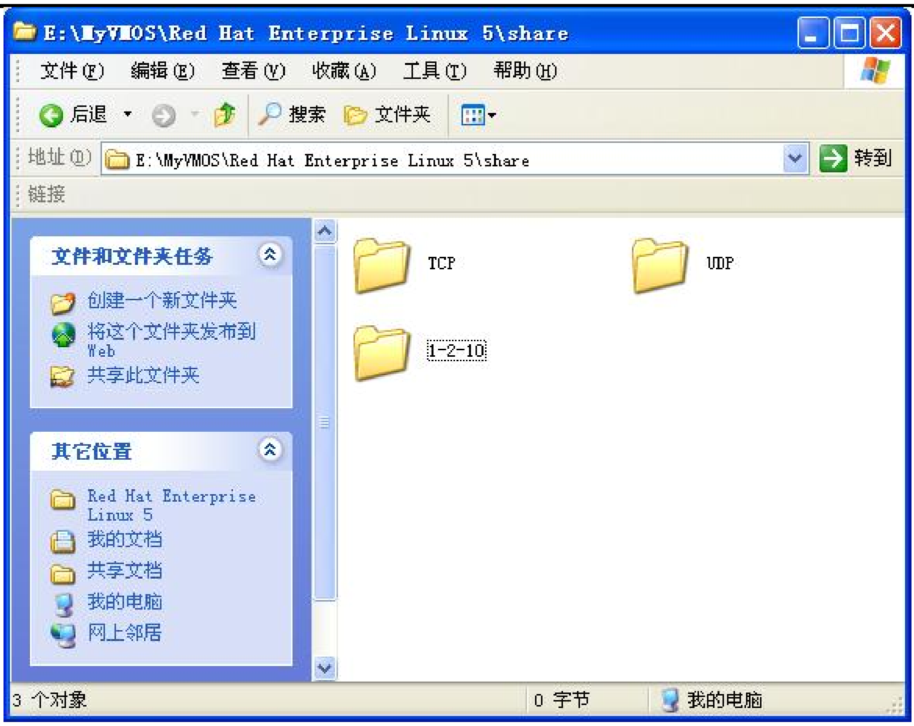
访问 Windows 系统中共享文件的步骤如下:

Step1: 在 Windows 中设置共享文件夹"share"(示例)

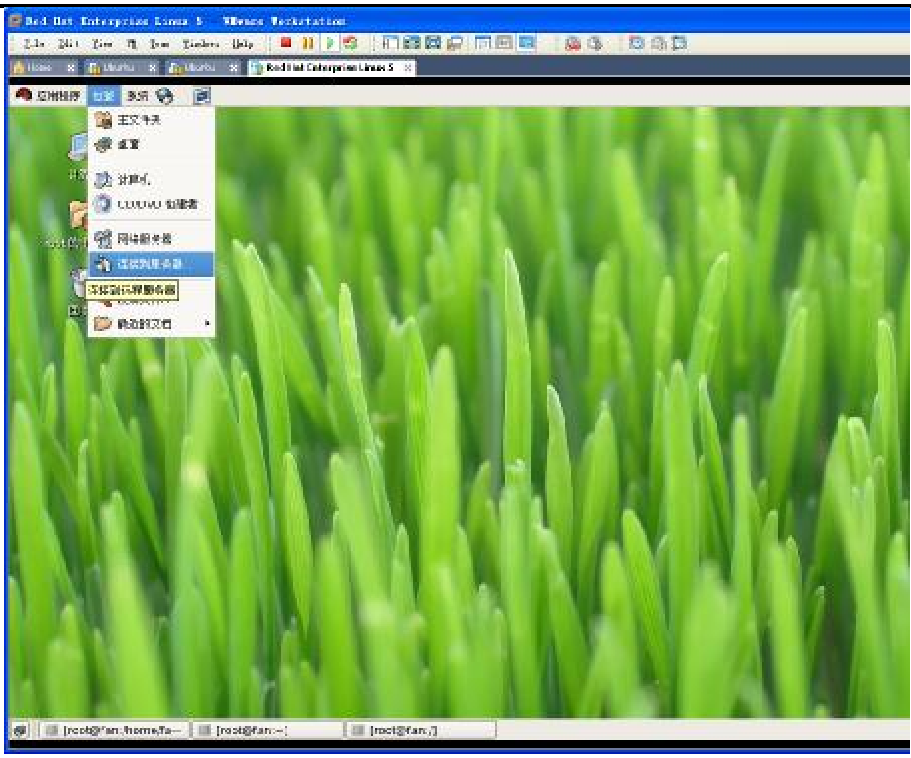


E:\MyVMOS\Red Hat Enterprise Linux 5
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) E:\MyVMOS\Red Hat Enterprise Linux 5
链接
文件和文件夹任务
重命名这个文件夹
移动这个文件夹
复制这个文件夹
将这个文件夹发布到Web
共享此文件夹
以电子邮件形式发送该文件夹内的文件
删除这个文件夹
其它位置
MyVMOS
我的文档
share
选定 1 个对象
share 属性
常规 共享 自定义
本地共享和安全
如果只跟这台计算机的其他用户共享这个文件夹，请将其拖动到共享文档文件夹。
要将这个文件夹及其子文件夹设为专用(只有您可以访问)，请选择以下复选框。
将这个文件夹设为专用(M)
网络共享和安全
要跟网络用户和这台计算机的其他用户共享这个文件夹，请选择下面的第一个复选框，键入一个共享名。
在网络上共享这个文件夹(S)
共享名(H): share
允许网络用户更改我的文件(W)
了解共享和安全的更多信息。
Windows 防火墙配置为允许此文件夹与网络上其他计算机共享。
查看 Windows 防火墙设置
确定 取消 应用(A)

Share 文件的内容:



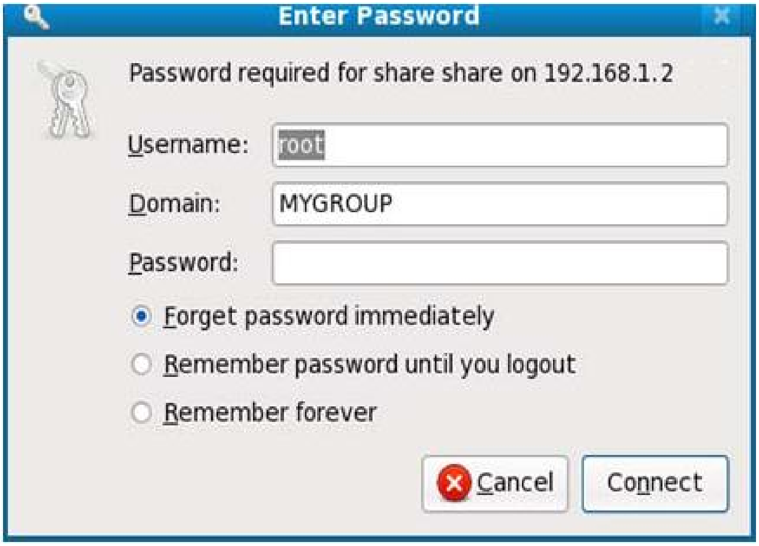
Step2: 在 RHEL5 系统中如图操作:



在打开的窗口中进行如图配置:



点"连接"，会出现如下提示窗口：



不必理会，直接点"Connect"即可，就可以看到 Windows 共享文件中的内容了，在此你可以像操作其他目录一样来使用它。



如果你想在命令行使用这个目录，可以这样操作：

说明：在控制台下，TAB 键是一个很好使用的小技巧。

5.3 建立交叉编译环境

在 Linux 平台下，要为开发板编译内核，图形界面 Qtopia，bootloader，还有其他一些应用程序，均需要交叉编译工具链。

之前的系统，要使用不同的编译器版本才能正常编译各个部分，因此要在开发过程不断切换设置，这十分不利于初学者使用，也降低了开发的效率；自从 Linux-2.6.29 开始(本开发板所配内核已为最新的 Linux-2.6.32.2)，我们把交叉编译器统一为 arm-linux-gcc-4.3.2，下面是它的安装设置步骤。

Step1: 将光盘目录**\Linux 源代码和工具**中的 arm-linux-gcc-4.3.2.tgz 复制到某个目录下如 tmp\，然后进入到该目录，执行解压命令：

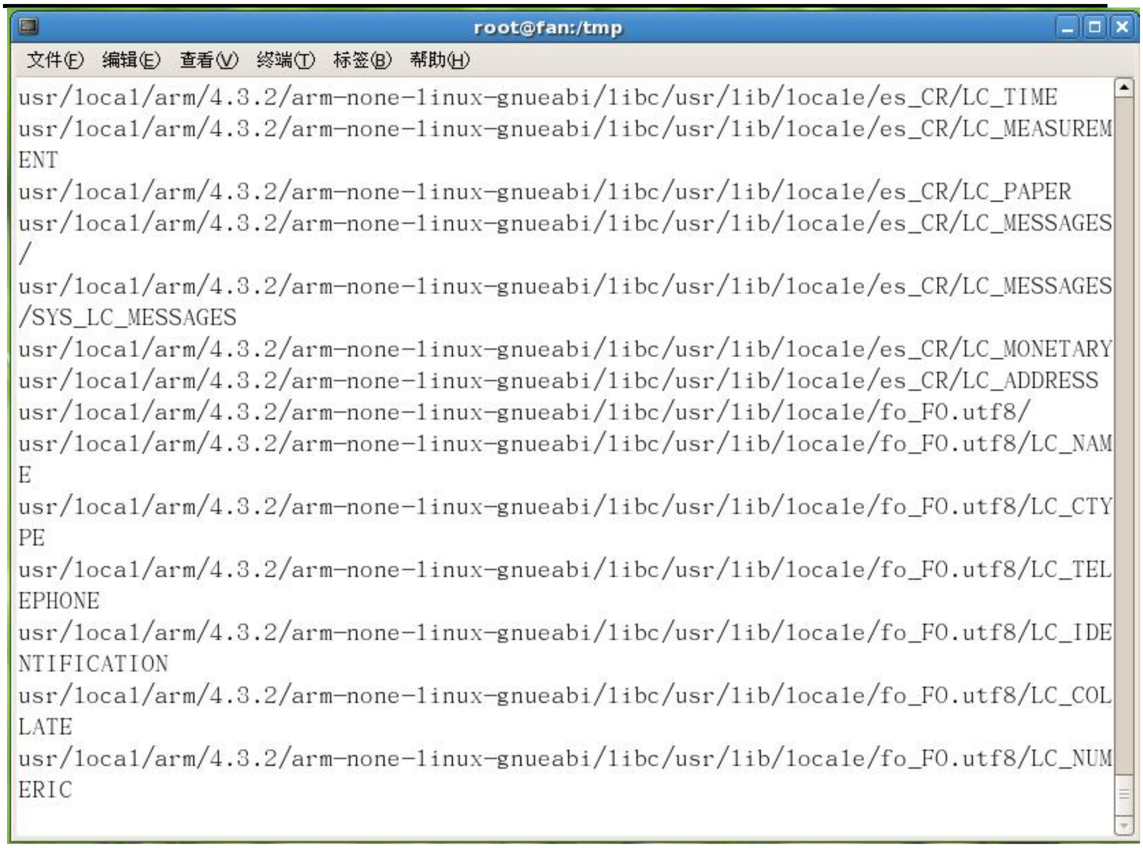
#cd \tmp

#tar xvzf arm-linux-gcc-4.3.2.tgz –C /

注意：C 后面有个空格，并且 C 是大写的，它是英文单词"Change"的第一个字母，在此是改变目录的意思。

执行该命令，将把 arm-linux-gcc 安装到/usr/loca/arm/4.3.2 目录。

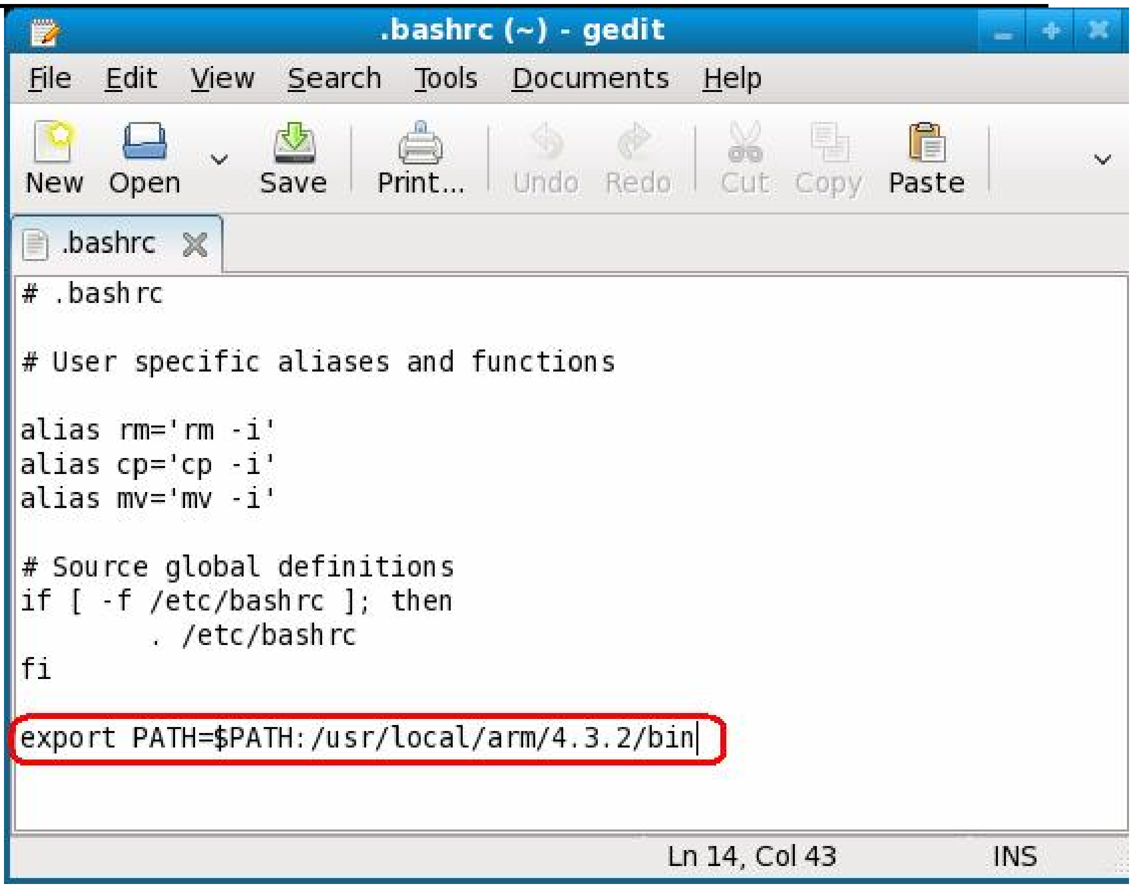
解压过程如图所示：



Step2: 把编译器路径加入系统环境变量，运行命令

#gedit /root/.bashrc

编辑/root/.bashrc 文件，在最后一行 export PATH=$PATH:/usr/local/arm/4.3.2/bin。如图，保存退出。



重新打开终端(不必重启机器)，使以上设置生效，在命令行输入 **arm-linux-gcc –v**，会出现如下信息，这说明交叉编译环境已经成功安装:

```
[root@fan /]# arm-linux-gcc-4.3.2 -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure --build
=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --en
able-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --wit
h-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers
=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72' --wit
h-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/
opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --
with-build-sysroot=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnue
abi/libc --with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-ar
m-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-resp
in/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --
disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scra
tch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-tim
e-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
[root@fan /]#
```

5.4 解压安装源代码及其他工具

本小节将解压安装开发学习过程所用到的全部源代码以及其他一些小工具，这包括:

- Linux 内核源代码
- 嵌入式图形界面 Qropia 源代码(分为 x86 和 arm 平台两个版本)
- busybox-1.13 源代码
- Linux 编程示例源代码(均为友善之臂自主开发并开放)
- 用以启动 Linux 的 bootloader 之 vboot
- 其他开源的 bootloader(适用于 Linux)
- 其他开源软件源代码，如 boa(web server), madplay(一个命令行 mp3 播放器)
- 目标文件系统目录
- 目标文件系统映象制作工具 mkyaffsimage
- 图形界面的 Linux logo 制作工具 logomaker

注意：所有的源代码和工具都是通过解压方式安装的，所有的源代码均使用统一的编译器 arm-linux-gcc-4.3.2 编译(见上一节)

下面是详细的解压安装过程，并有简要的介绍。

5.4.1 解压安装源代码

首先创建工作目录**/opt/Enjoylinux/GQ2440**，在命令行执行 mkdir –p /opt/Enjoylinux/GQ2440 ，如图，后面步骤的所有源代码都会解压安装到此目录中：



(1)准备好 Linux 源代码包

在 RHEL5 系统中/tmp 目录中创建一个临时目录/tmp/linux

#mkdir /tmp/linux

把光盘中**\Linux 源代码和工具**目录中的所有文件都复制到/tmp/linux 目录中

说明：这样做是为了统一下面的操作步骤，其实你可以使用其他目录，也可以直接从光盘解压安装。

(2)解压安装 Linux 内核源代码

在工作目录**/opt/Enjoylinux/GQ2440** 中执行：

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/linux-2.6.32.2-GQ2440.tar.gz

将创建生成 linux-2.6.32.2 目录，里面包含了完整的 Linux-2.6.32.2 内核源代码。

(3)解压安装嵌入式图形系统 qtopia 源代码

在工作目录**/opt/Enjoylinux/GQ2440** 中执行：

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/x86-qtopia.tgz

#tar xvzf /tmp/linux/arm-qtopia.tgz

将创建 x86-qtopia 和 arm-qtopia 两个目录，并内含相应的全部源代码。

说明：压缩文件名后面可能跟上发行日期，如 **arm-qtopia.tgz** 在光盘中可能是 **arm-qtopia-20100108.tar.gz** ，其中 **20100108** 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。和之前的 qtopia 源代码包不同，现在的源代码包不再区分 mouse(鼠标支持)和 tp(触摸屏支持)，此系统可以支持二者共存，因此只有一个源代码包，其中也包含了嵌入式浏览器 konquor 的源代码。

另外，为了方便用户学习开发使用，此源代码包相比 Qt 公司的原始版本已经打过补丁，并做了诸多改进，它们都是源代码方式，我们不再一一赘述，感兴趣者可自行比较。

(4)解压安装 busybox 源代码

Busybox 是一个轻型的 linux 命令工具集，在此使用的是 busybox-1.13.3 版本。用户可以从其官方网站下载最新版本(http://www.busybox.net)。

在工作目录**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/busybox-1.13.3-GQ2440.tar.gz

将创建 busybox-1.13.3 目录，内含相应版本的全部源代码。

说明：为了方便用户编译使用，我们做了一个缺省的配置文件 fa.config。

(5)解压安装 Linux 示例程序

在工作目录**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/examples.tgz

将创建 examples 目录，并包含初学 linux 编程代码示例。

(6)解压安装 vboot 源代码

为了实现自动适应支持 64M/128M GQ2440，我们专门为 Linux 系统设计了一个简易的 bootloader：vboot，而不再使用以前的 vivi。

在工作目录**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/vboot-GQ2440.tar.gz

将创建 vboot 目录，里面包含该 bootloader 的源代码和 Makefile 文件。

(7)解压安装其他其他开源 bootloader 源代码

除了 vboot，本开发板还提供了另外三种开源的 Bootloader(详见本手册第八章节)，其中 vivi 和 u-boot 是在 Linux 平台下设计编译的。

在工作目录**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/bootloader.tgz

将创建 bootloader 目录，里面包含 vivi 和 u-boot 两种 bootloader 的源代码。

说明：此处的 vivi 仅适用于 64M Nand Flash 的 GQ2440 板，u-boot 是由网友提供的，我们并没有使用过，对其功能和性能均不了解。

5.4.2 解压创建目标文件系统

在工作目录/**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/root_qtopia.tar.gz

将创建 root_qtopia 目录，该目录和目标板上使用的文件系统内容是完全一致的。

说明：这个根文件系统包含了完整的 qtopia 测试系统，最新的 busybox，还有常用的命令行工具等，它具有如下特性:

- 自动识别 NFS 启动或本地启动
- 可支持 USB 鼠标和触摸屏共存
- 自动识别所接的输出显示模块是否接了触摸屏，以判断在第一次开机使用时是否要进行校正。如果没有连接，会自动进入系统，使用鼠标即可；否则会先校正触摸屏。

5.4.3 解压安装必要实用工具

目标文件系统映象制作工具 mkyaffs2image

要把上一步中的 root_qtopia 目录烧写入目标板中使用， 就需要使用相应的 mkyaffs2image 工具了，它是一个命令行的程序，使用它可以把主机上的目标文件系统目录制作成一个映象文件，以烧写到开发板中。针对 64M 或 128M/256M/512M/1GB 的 GQ2440，分别有 2 套制作工具：mkyaffs2image 和 mkyaffs2image-128M。其中

mkyaffs2image 是制作适用于 64M 版本文件系统映象的工具， 它沿用了以前的名字；mkyaffs2image-128M 是制作适用于 128M/256M/512M/1GB 版本文件系统映象的工具，为了便于区分，我们把它命名为此。

在工作目录**/opt/Enjoylinux/GQ2440** 中执行:

#cd /opt/Enjoylinux/GQ2440

#tar xvzf /tmp/linux/mkyaffs2image.tgz –C /

注意: C 是大写的，C 后面有个空格，C 是改变解压安装目录的意思。

说明: 以前的内核系统支持的是 yaffs 文件系统，现在使用的是 yaffs2 文件系统，因此需要不同的制作工具，我们在此把它称为 mkyaffs2image，按照上面的命令解压后它会被安装到/usr/sbin 目录下，并产生 2 个文件：mkyaffs2image 和 mkyaffs2image-128M。

5.5 配置网络文件系统 NFS 服务

如果您已经按照以上章节介绍的方法完全安装好了 RHEL5，则 NFS 相关软件都已经缺省安装好了，请按照以下步骤建立和配置 NFS 服务。

5.5.1 设置共享目录

注意: 要使用共享目录，必须先按照5.4.2 章节解压安装好root_qtopia 目标板文件系统包。

(1)设置共享目录

运行命令

#gedit /etc/exports

编辑 nfs 服务的配置文件(注意：第一次打开时该文件是空的)，添加以下内容:

/opt/Enjoylinux/GQ2440　*(rw,sync,no_root_squash)

其中:

/opt/Enjoylinux/GQ2440 表示 nfs 共享目录，它可以作为开发板的根文件系统通过 nfs 挂接;

***** 表示所有的客户机都可以挂接此目录

rw 表示挂接此目录的客户机对该目录有读写的权力

no_root_squash 表示允许挂接此目录的客户机享有该主机的 root 身份。

5.5.2 启动 NFS 服务

可以通过命令行和图形界面两种方式启动 NFS 服务，我们建立 NFS 服务的目的是通
过网络对外提供目录共享服务，但默认安装的RHEL5系统开启了防火墙，这会导致NFS 服务无法正常使用。因此先关闭防火墙，在命令行输入“lokkit”命令，打开防火期设置界面：



选择其中(*)关闭，然后选择“确定”退出，这样就永久的关闭了防火墙。

下面是启动 NFS 服务的方法和步骤：

(1)通过命令启动和停止 nfs 服务在命令行下运行:

#/etc/init.d/nfs start

这将启动 nfs 服务，可以输入以下命令检验 nfs 该服务是否启动。

mount -t nfs localhost: /opt/Enjoylinux/GQ2440/root_qtopia　/mnt/

如果没有出错信息，您将可以看到**/mnt** 目录中的内容和**/opt/Enjoylinux/GQ2440/root_qtopia** 是一致的。

使用这个命令可以停止 nfs 服务:

#/etc/init.d/nfs stop

(2)通过图形界面启动 NFS 服务

为了在每次开机时系统都自动启动该服务，可以输入

serviceconf

打开系统服务配置窗口，在左侧一栏找到 nfs 服务选项框，并选中它，然后点其前面的方框划勾，如图。

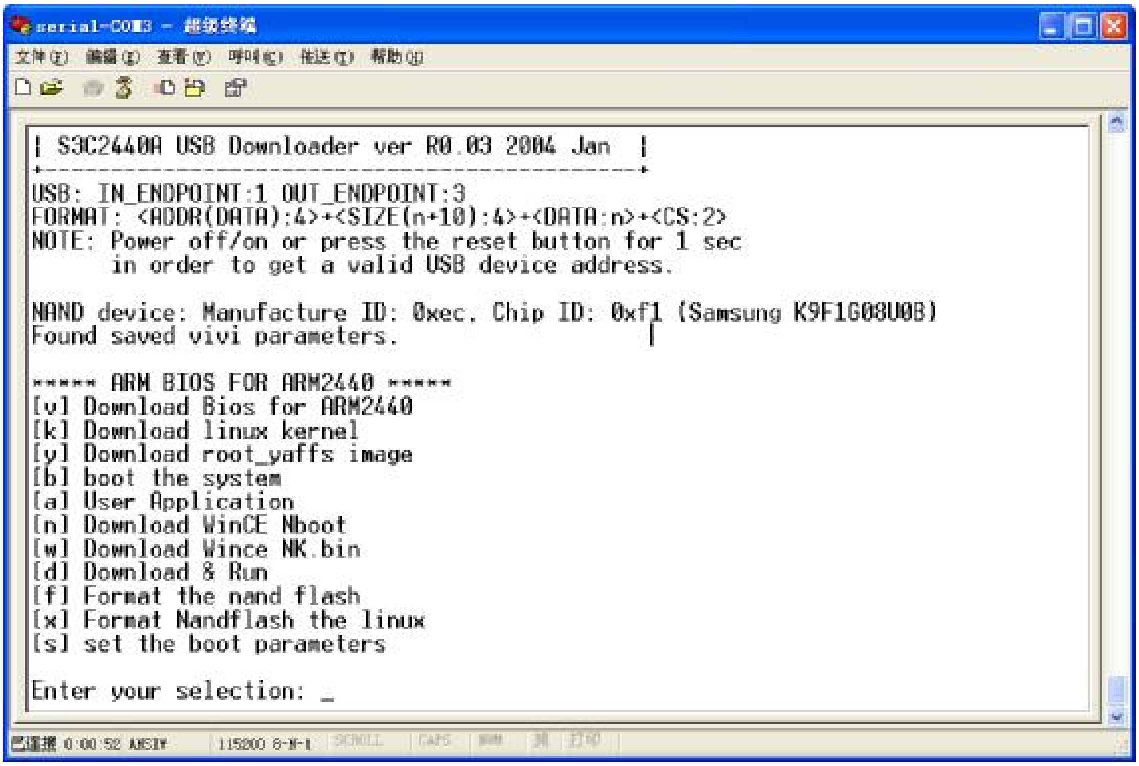


5.5.3 通过 NFS 启动系统

当 NFS 服务设置好并启动后，我们就可以把 NFS 作为根文件系统来启动开发板了。通过使用 NFS 作为根文件系统，开发板的“硬盘”就可以变得很大，因为您使用的是主机的硬盘，这是使用 Linux 作为开发经常使用的方法，

设置目标板启动模式为 Nand Flash 启动，连接好电源，串口线，网线；打开串口终端，

在开机或者复位的时候按下开发板上的 K1-K6 任意按键（最好在启动前就按着），这样我们就进入了 vivi 模式，如图：



输入以下命令：

Supervivi> param set linux_cmd_line "console=ttySAC0 root=/dev/nfs nfsroot=192.168.1.5:/opt/Enjoylinux/GQ2440/root_qtopia

ip=192.168.1.230:192.168.1.5:192.168.1.5:255.255.255.0:GQ2440:eth0:on"

其中，param set linux_cmd_line 是设置启动 linux 时的命令参数。其各参数的含义如下：

nfsroot 是 nfs 服务器主机的 IP 地址，即 pc 机。

"ip="后面：

第一项(192.168.1.230)是目标板的临时 IP(注意不要和局域网内其他 IP 冲突);

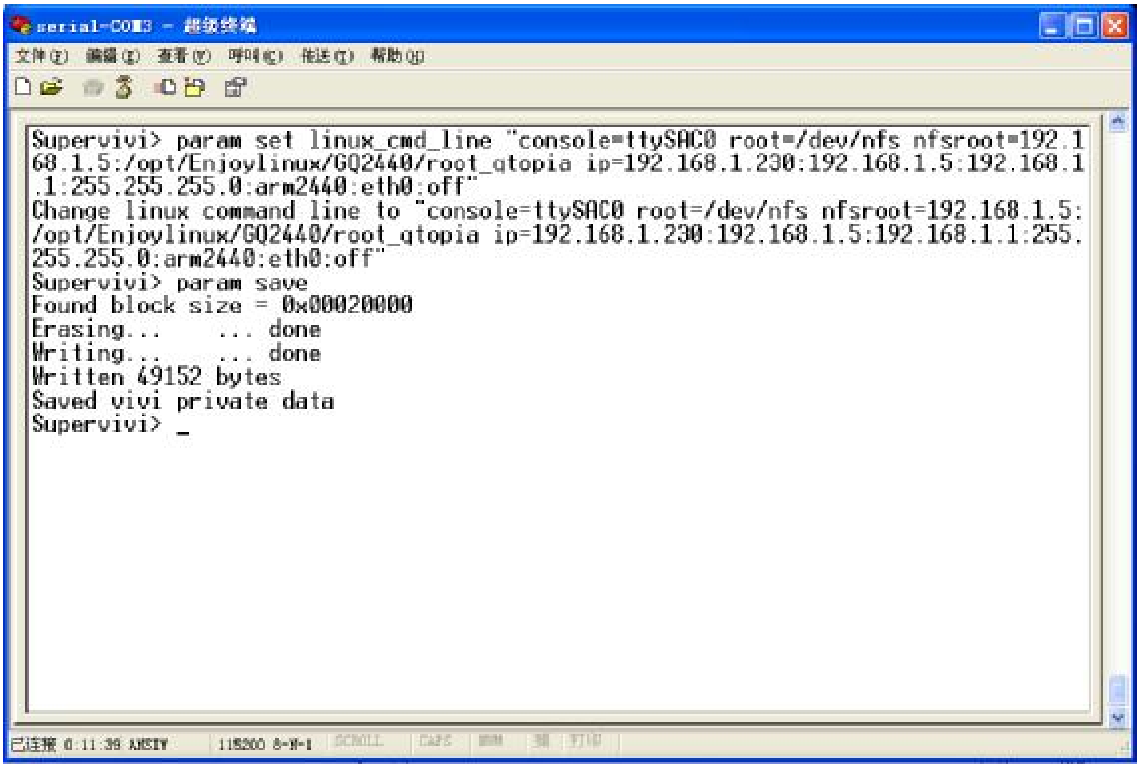
第二项(192.168.1.5)是 nfs 服务器的 IP;

第三项(192.168.1.1)是目标板上网关(GW)的设置;

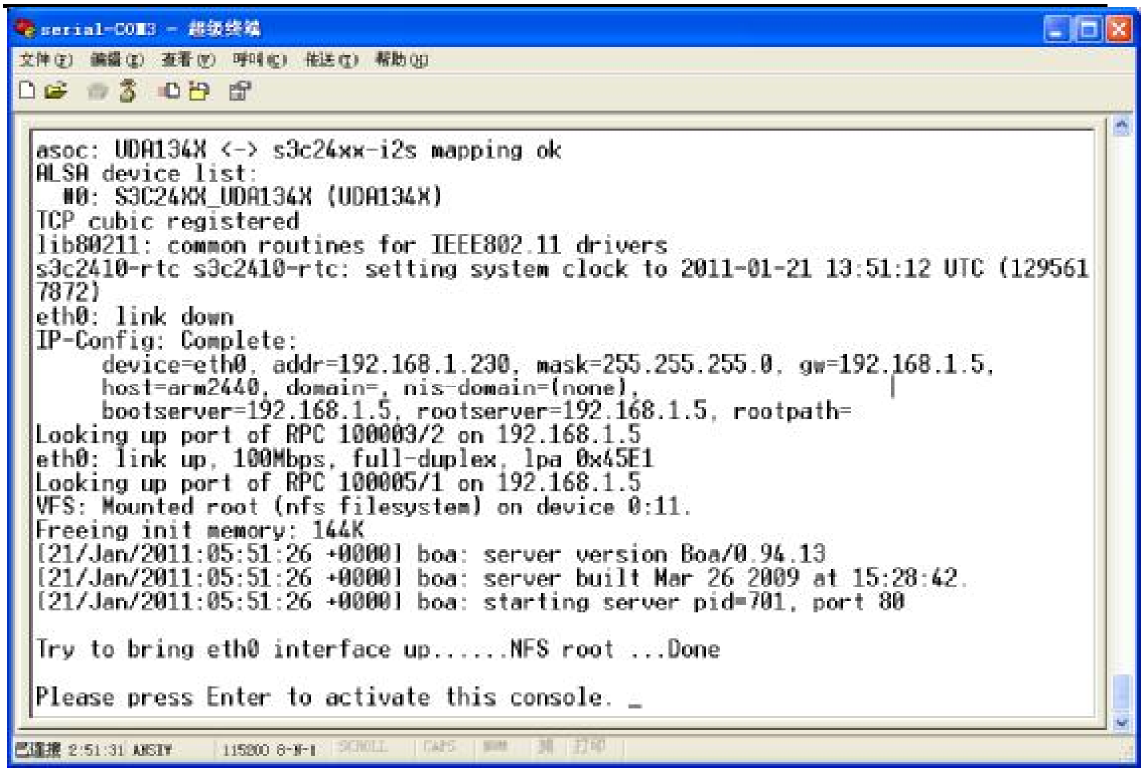
第四项(255.255.255.0)是子网掩码;

第五项是开发主机的名字(一般无关紧要，可随便填写)

eth0 是网卡设备的名称。



然后输入 boot，按回车就可以通过 nfs 启动系统了。



```
asoc: UDA134X <-> s3c24xx-i2s mapping ok
ALSA device list:
  #0: S3C24XX_UDA134X (UDA134X)
TCP cubic registered
lib80211: common routines for IEEE802.11 drivers
s3c2410-rtc s3c2410-rtc: setting system clock to 2011-01-21 13:51:12 UTC (129561
7872)
eth0: link down
IP-Config: Complete:
     device=eth0, addr=192.168.1.230, mask=255.255.255.0, gw=192.168.1.5,
     host=arm2440, domain=, nis-domain=(none),
     bootserver=192.168.1.5, rootserver=192.168.1.5, rootpath=
Looking up port of RPC 100003/2 on 192.168.1.5
eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
Looking up port of RPC 100005/1 on 192.168.1.5
VFS: Mounted root (nfs filesystem) on device 0:11.
Freeing init memory: 144K
[21/Jan/2011:05:51:26 +0000] boa: server version Boa/0.94.13
[21/Jan/2011:05:51:26 +0000] boa: server built Mar 26 2009 at 15:28:42.
[21/Jan/2011:05:51:26 +0000] boa: starting server pid=701, port 80

Try to bring eth0 interface up......NFS root ...Done

Please press Enter to activate this console. _
```

第6章 定制 Linux 内核及制作文件系统

Linux 内核看似很庞大，其实初学者以及致力于应用开发的用户，不必学习之初就一头钻进浩如烟海的内核代码中问剑；但熟练配置内核的一些常用选项，并编译出来下载到开发板中运行试用，是你迈向 Linux 系统顶峰的必行之路。

本小节中所有步骤内容均不涉及代码编写，但请初学者不要忽视它们；学习 Linux 不像单片机系统，你不必从“零”代码开始，一切先从学会配置、编译、下载运行开始。下面操作均基于 RHEL5 平台操作。

注意：本开发板所配最新内核版本为 Linux-2.6.32.2

6.1 使用缺省配置文件配置和编译内核

为了方便用户能够编译出和光盘烧写文件完全一致的内核，我们针对不同的 LCD 输出分别做了相应的内核配置文件：

config_GQ2440_s35 – 适用于三星 3.5” LCD 的内核配置文件

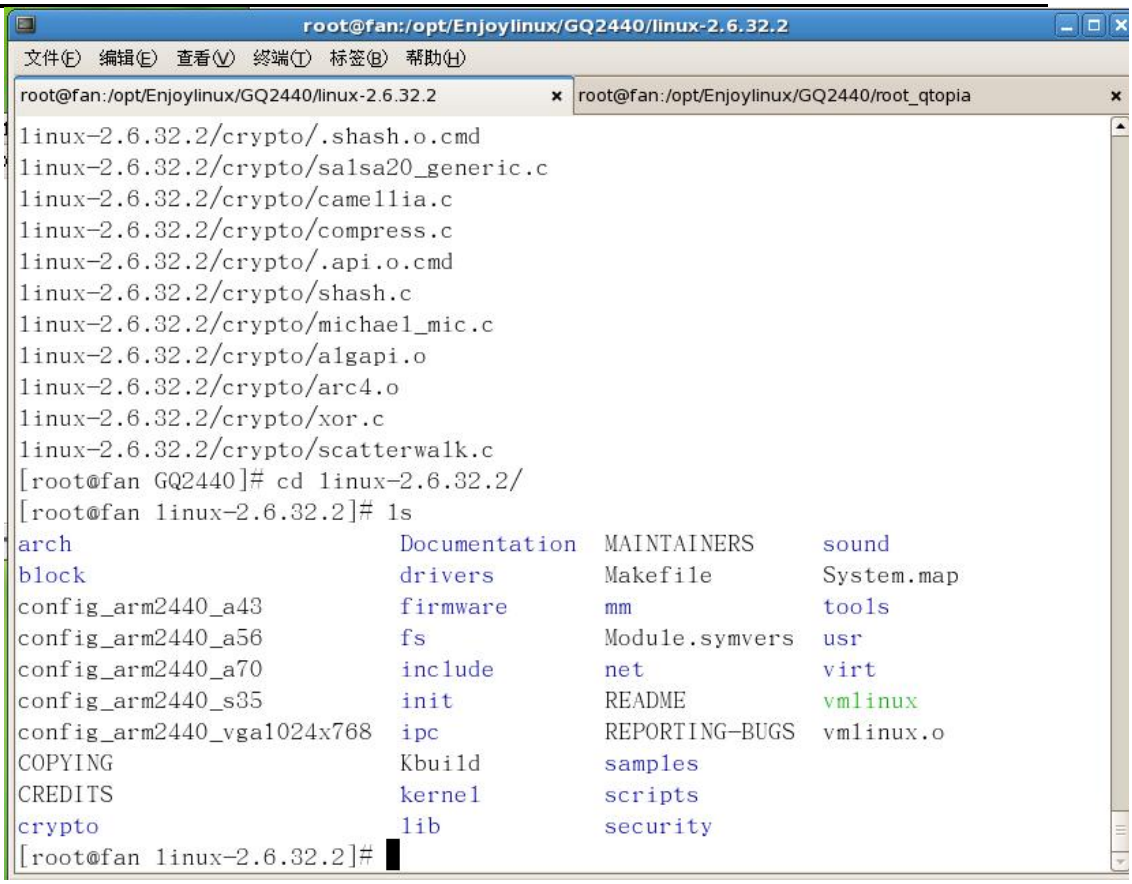
config_GQ2440_a43 – 适用于群创 4.3” LCD 的内核配置文件

config_GQ2440_a56 – 适用于群创 5.6” LCD 的内核配置文件

config_GQ2440_a70 – 适用于群创 7” LCD 的内核配置文件

config_GQ2440_vga1024x768 – 适用于 VGA 显示输出(分辨率 1024x768）模块的内核配置文件

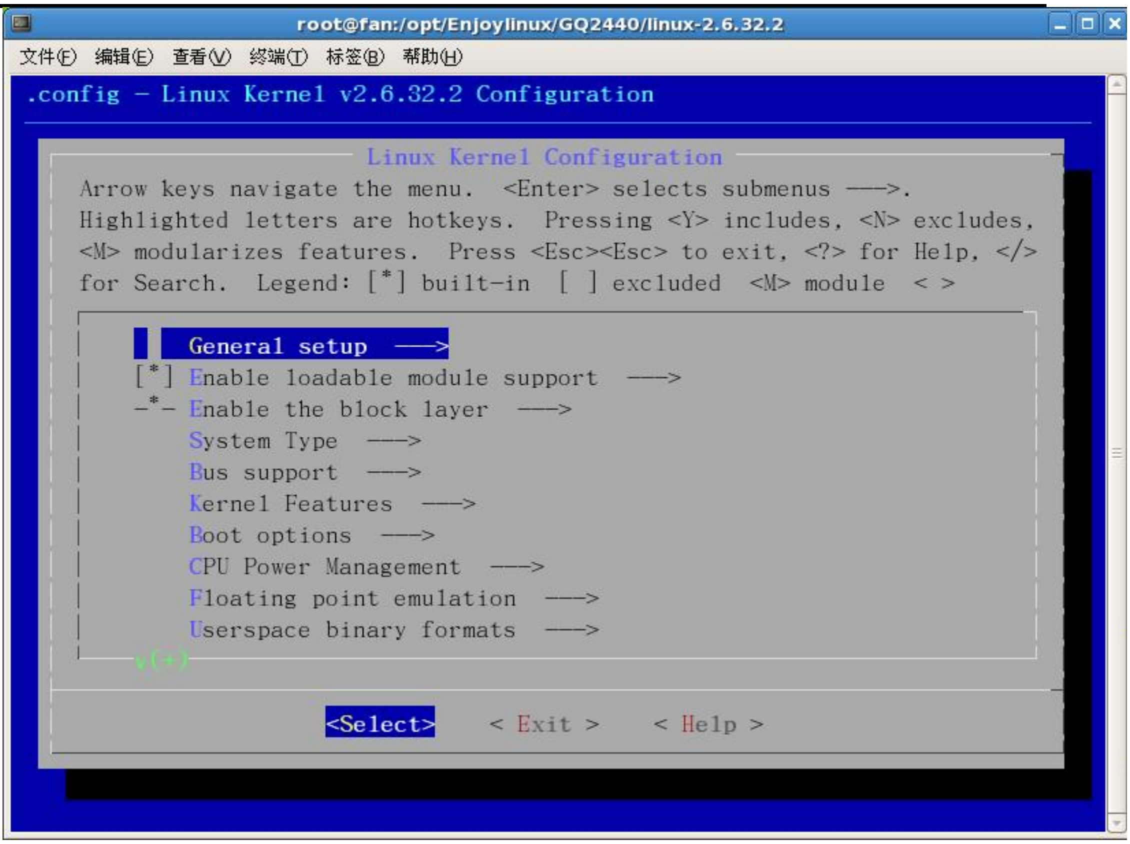
在内核目录中可以使用 ls 命令看到它们的存在:



执行以下命令来使用缺省配置文件 config_s35

#cp config_GQ2440_s35 .config *;注意：s35 后面有个空格，然后有个“.”开头的 config*

然后执行“**make menuconfig**”，出现配置内核界面：



这时不用做任何更改，在主菜单里选择<Exit>退出，这样做是为了生成相应配置的头文件。

输入以下命令，开始编译内核：

#make zImage

```
root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
  GEN     .version
  CHK     include/linux/compile.h
  UPD     include/linux/compile.h
  CC      init/version.o
  LD      init/built-in.o
  LD      .tmp_vmlinux1
  KSYM    .tmp_kallsyms1.S
  AS      .tmp_kallsyms1.o
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  GZIP    arch/arm/boot/compressed/piggy.gz
  AS      arch/arm/boot/compressed/piggy.o
  CC      arch/arm/boot/compressed/misc.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
[root@fan linux-2.6.32.2]#
```

您可以使用第三章节介绍的方法把 zImage 下载到开发板测试。

6.2 使用缺省配置文件配置和编译内核

说明：解压内核源代码(linux-2.6.32.2 开头的 tgz 压缩文件)可以找到如下驱动，GQ2440 提供基于 linux-2.6.32.2 内核 100%完全可以使用的驱动源代码，绝无库文件，敬请放心使用。

(1)DM9000 网卡驱动

Linux-2.6.32.2/drivers/net/dm9000.c

(2)串口(包括三个串口驱动 0,1,2，对应设备名/dev/ttySAC0,1,2)

Linux-2.6.32.2/drivers/serial/s3c2440.c

(3)实时时钟 RTC 驱动

Linux-2.6.32.2/drivers/rtc/rtc-s3c.c

(4)LED 驱动

Linux-2.6.32.2/drivers/char/GQ2440_leds.c

(5)按键驱动

Linux-2.6.32.2/drivers/char/GQ2440_buttons.c

(6)触摸屏驱动

Linux-2.6.32.2/drivers/input/touchscreen/s3c2410_ts.c

(7)yaffs2 文件系统源代码目录

Linux-2.6.32.2/fs/yaffs2

(8)USB 鼠标、键盘源代码

Linux-2.6.32.2/drivers/usb/hid

(9)SD/MMC 卡驱动源代码目录(支持高速最大容量 32G SD 卡)

Linux-2.6.32.2/drivers/mmc

(10)Nand Flash 驱动

Linux-2.6.32.2/drivers/mtd/nand

(11)UDA1341 音频驱动目录

Linux-2.6.32.2/sound/soc/s3c24xx

(12)LCD 驱动

Linux-2.6.32.2/drivers/video/s3c2410fb.c

(13)优盘支持驱动

Linux-2.6.32.2/drivers/usb/storage

(14)万能 USB 摄像头驱动

Linux-2.6.32.2/drivers/media/video/gspca

(15)I2C-EEPROM 驱动

inux-2.6.32.2/drivers/i2c

(16)背光驱动

Linux-2.6.32.2/drivers/video/GQ2440_backlight.c

(17)PWM 控制蜂鸣器驱动

Linux-2.6.32.2/drivers/char/GQ2440_pwm.c

(18)看门狗驱动

Linux-2.6.32.2/drivers/watchdog/s3c2410_wdt.c

(19)AD 转换驱动

Linux-2.6.32.2/drivers/char/GQ2440_ad.c

(20)CMOS 摄像头驱动

Linux-2.6.32.2/drivers/media/video/s3c2440camif.c

(21)USB 无线网卡驱动(型号：TL-WN321G+)

Linux-2.6.32.2/drivers/net/wireless/rt2x00

(22)USB 转串口驱动

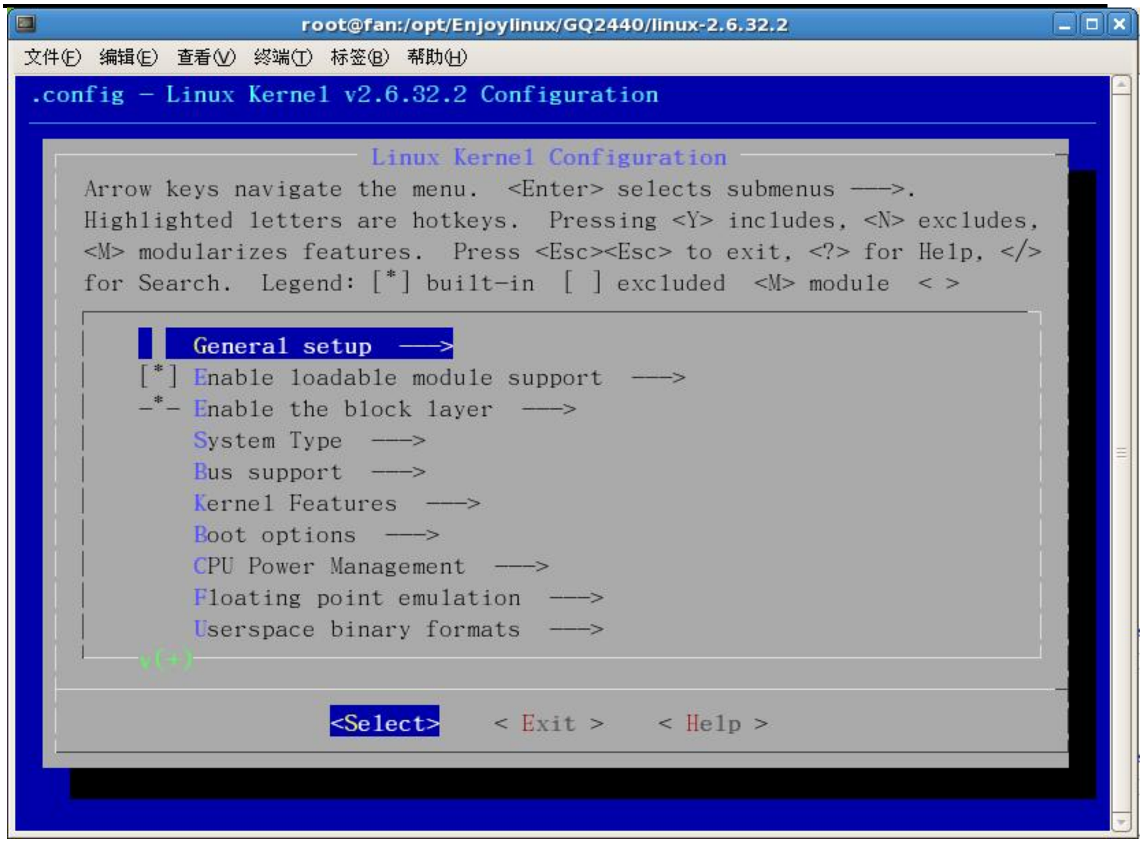
Linux-2.6.32.2/drivers/usb/serial/pl2302.c

6.3 手工定制 Linux 内核

上面我们使用缺省文件配置和编译了内核，其实 Linux 内核的配置选项有很多，下面我们就常见的一些选项分别予以图解，帮助你尽快熟悉内核配置，以便定制自己需要的内核。

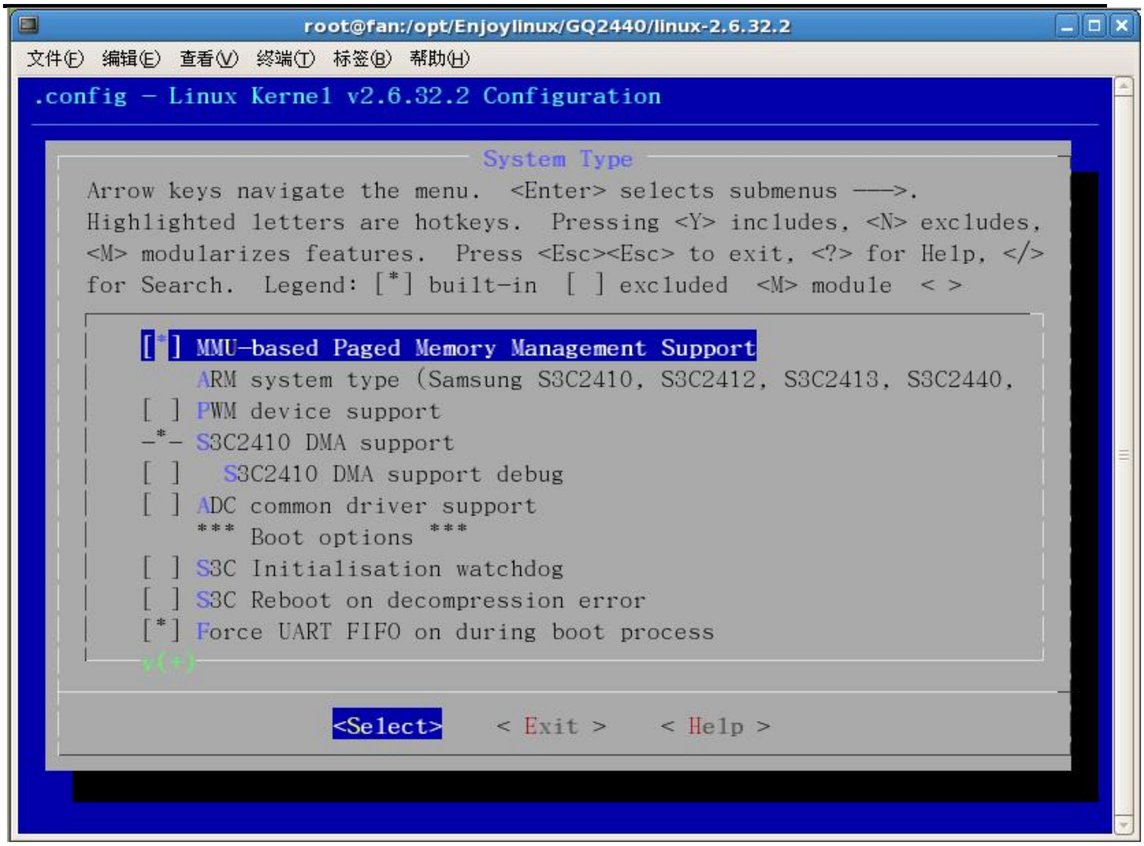
注意：为了方便你尽快熟悉各个内核配置选项，请务必按照 6.1 介绍的方法先加载一个缺省的配置文件如 **config_GQ2440_s35** ，否则下面的选项有可能是不会出现的。

运行 **make menuconfig** 后，进入内核配置主菜单



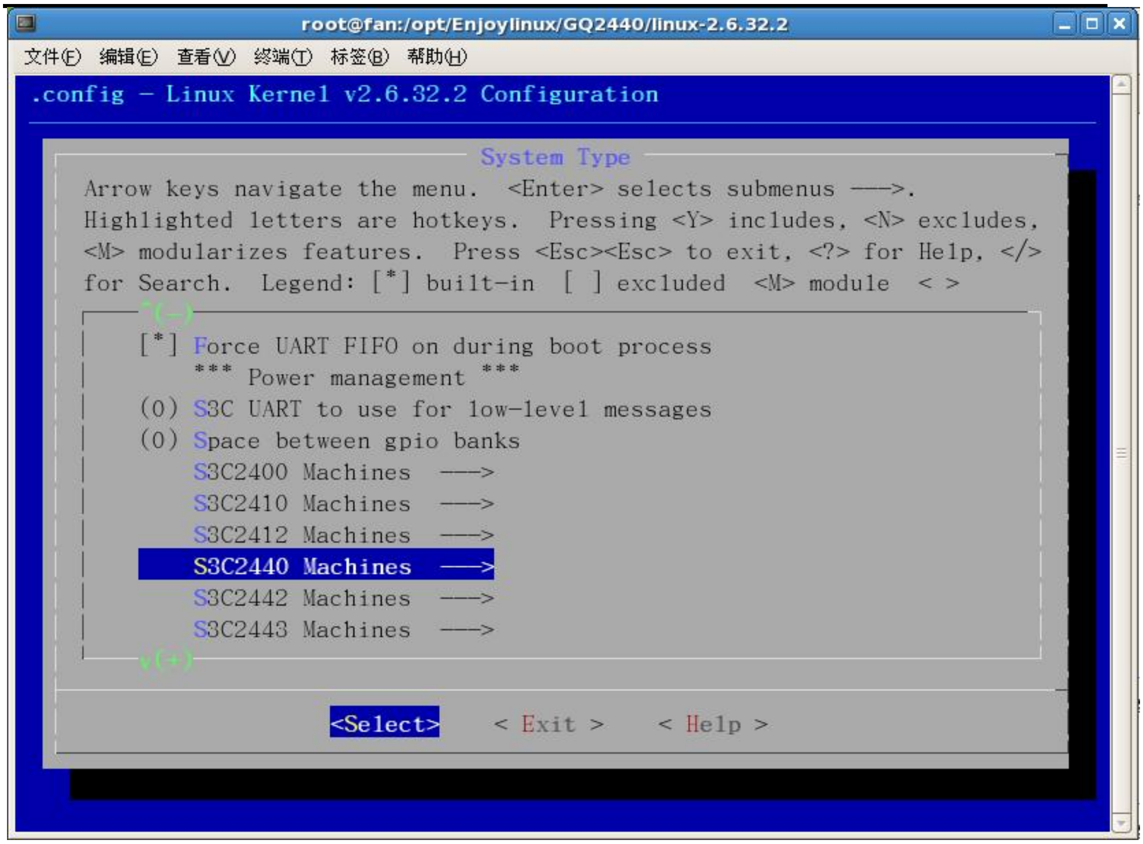
6.3.1 配置 CPU 平台选项

在主菜单里面，选择 System Type，按回车进入

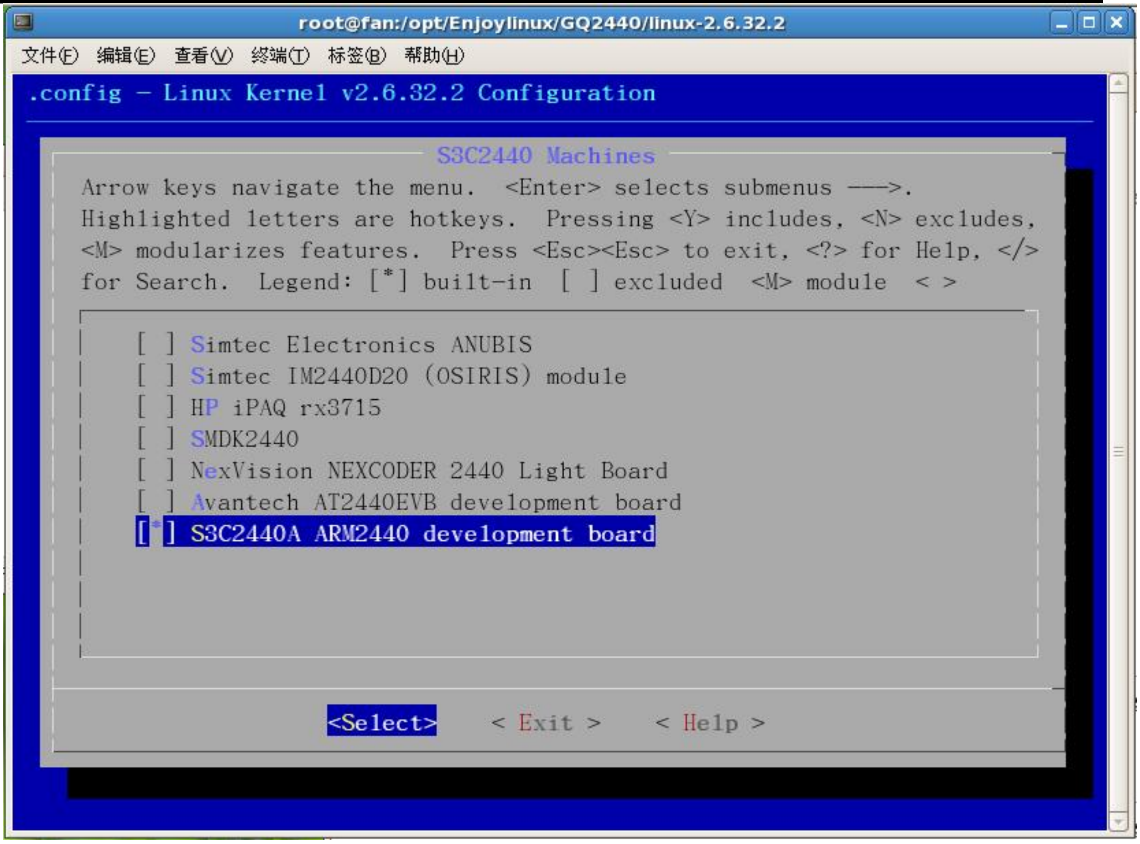


可以看到，系统大部分使用了标注了 S3C2410 的选项，这是因为 S3C2410 和 S3C2440 的很多寄存器地址等地址和设置是完全相同的。

如果您要选择板级选项，使用上下方向控制键一直找到 S3C2440 机器平台选项，可以进入 S3C2440 Machines 子菜单



可以看到里面有很多常见的使用 S3C2440 的目标板平台选项，在此选“S3C2440A GQ2440 development board”，如图

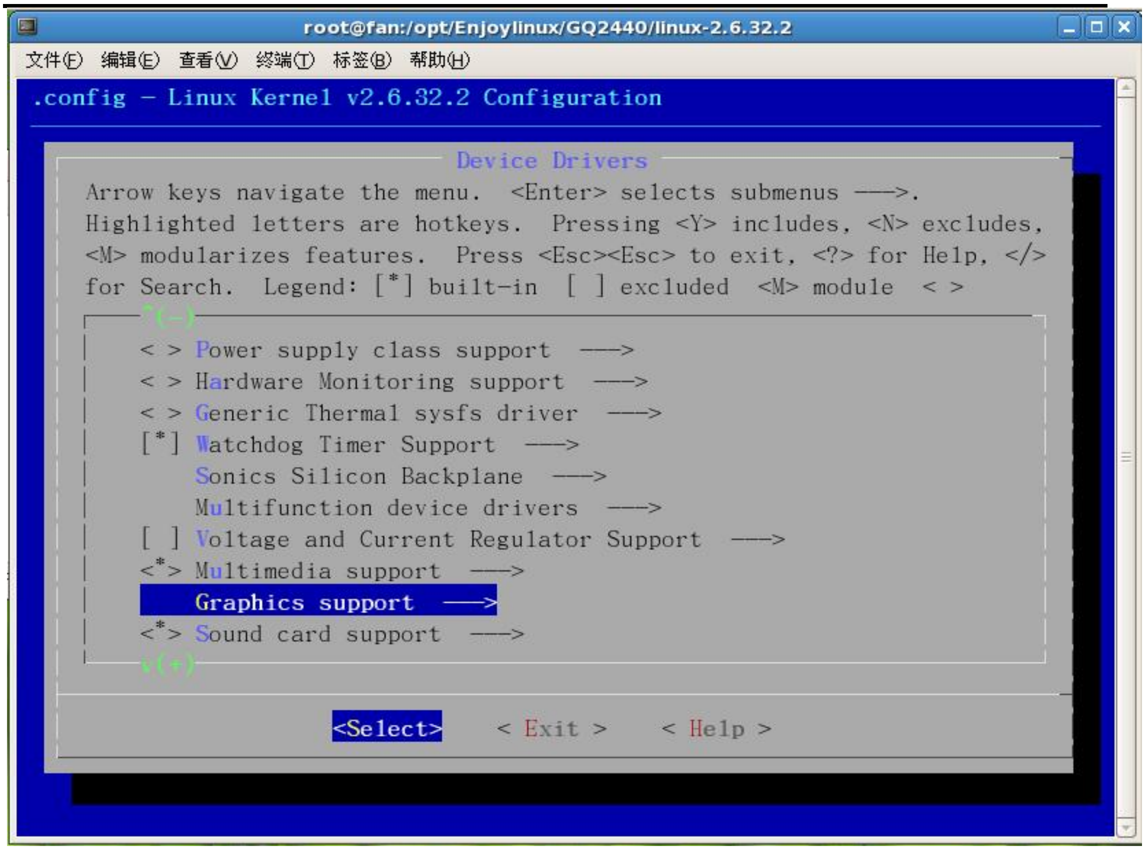


它们分别对 应 于 arch/arm/mach-s3c2440/mach-* 开头的文件 ， 在此对应 mach-arm 2440.c。另外，在这个文件中，还会用到一个机器码 MACH_TYPE，该机器码的定义文件为 arch/arm/tools/mach-types，我们开发板的机器码为 1999，它还对应于 vivi 源代码中 include/platform/smdk2440.h 文件的 MACH_TYPE

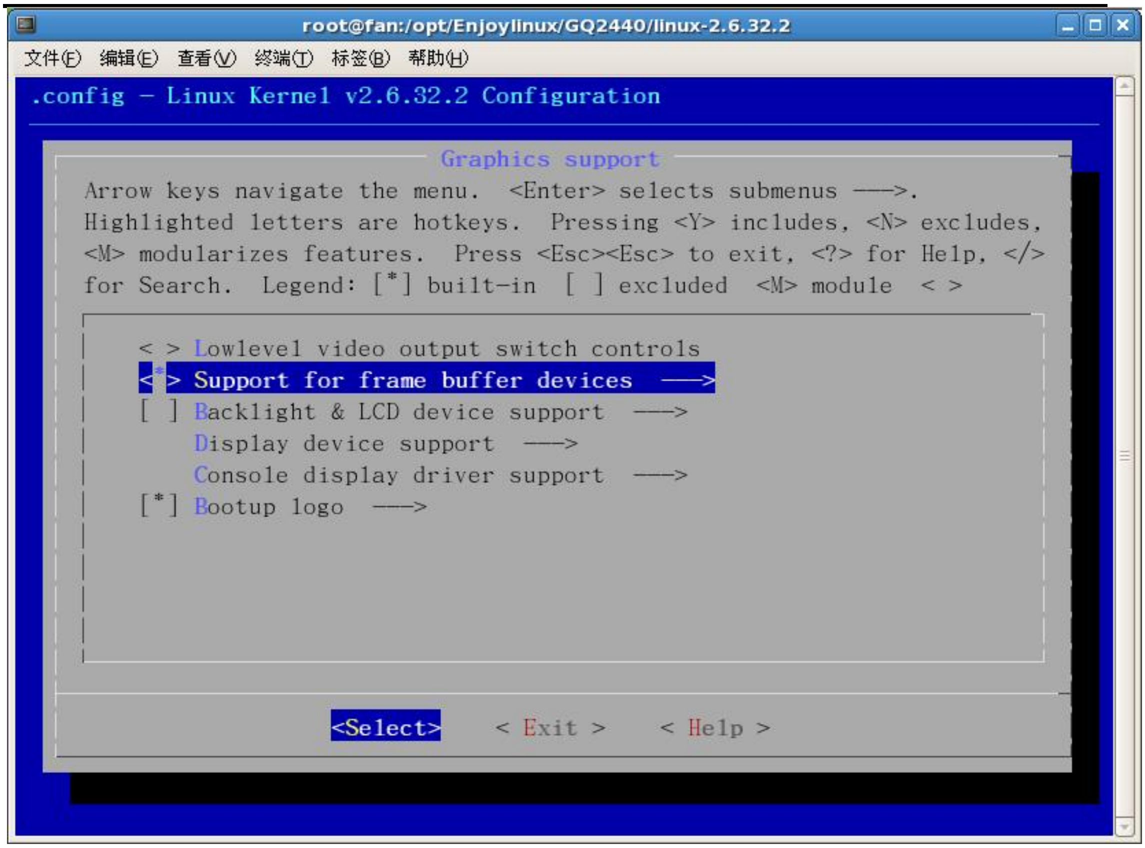
```
root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@fan ~]# cd /opt/Enjoylinux/GQ2440/
[root@fan GQ2440]# cd linux-2.6.32.2/
[root@fan linux-2.6.32.2]# cd arch/arm/mach-s3c2440/
built-in.o          .dsc.o.cmd            mach-nexcoder.c
.built-in.o.cmd     irq.c                 mach-osiris.c
clock.c             irq.o                 mach-rx3715.c
clock.o             .irq.o.cmd            mach-smdk2440.c
.clock.o.cmd        Kconfig               Makefile
dma.c               mach-anubis.c         s3c2440.c
dma.o               mach-arm2440.c        s3c2440.o
.dma.o.cmd          mach-arm2440.o        .s3c2440.o.cmd
dsc.c               .mach-arm2440.o.cmd
dsc.o               mach-at2440evb.c
[root@fan linux-2.6.32.2]#
```

6.3.2 配置各个尺寸的 LCD 驱动以及背光控制支持

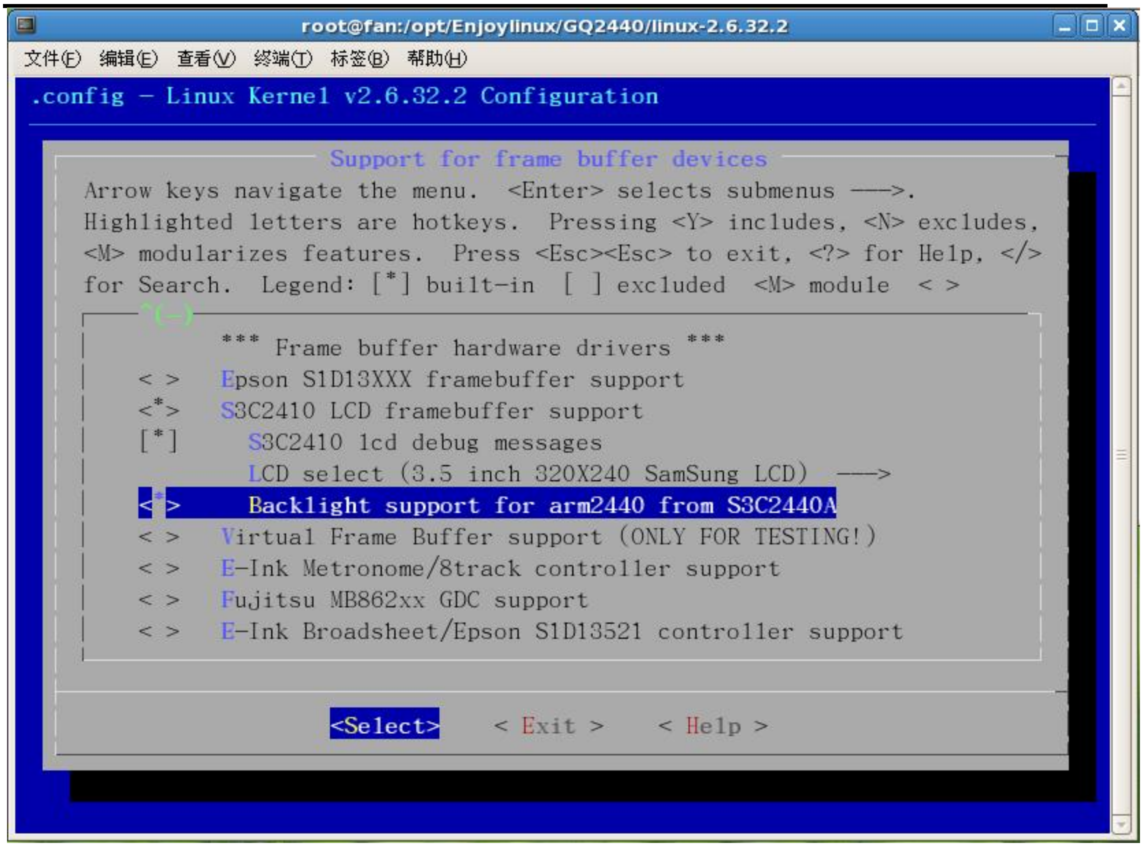
在主菜单里面，选择 Device Drivers，按回车进入，并找到如图选项，按回车进入：



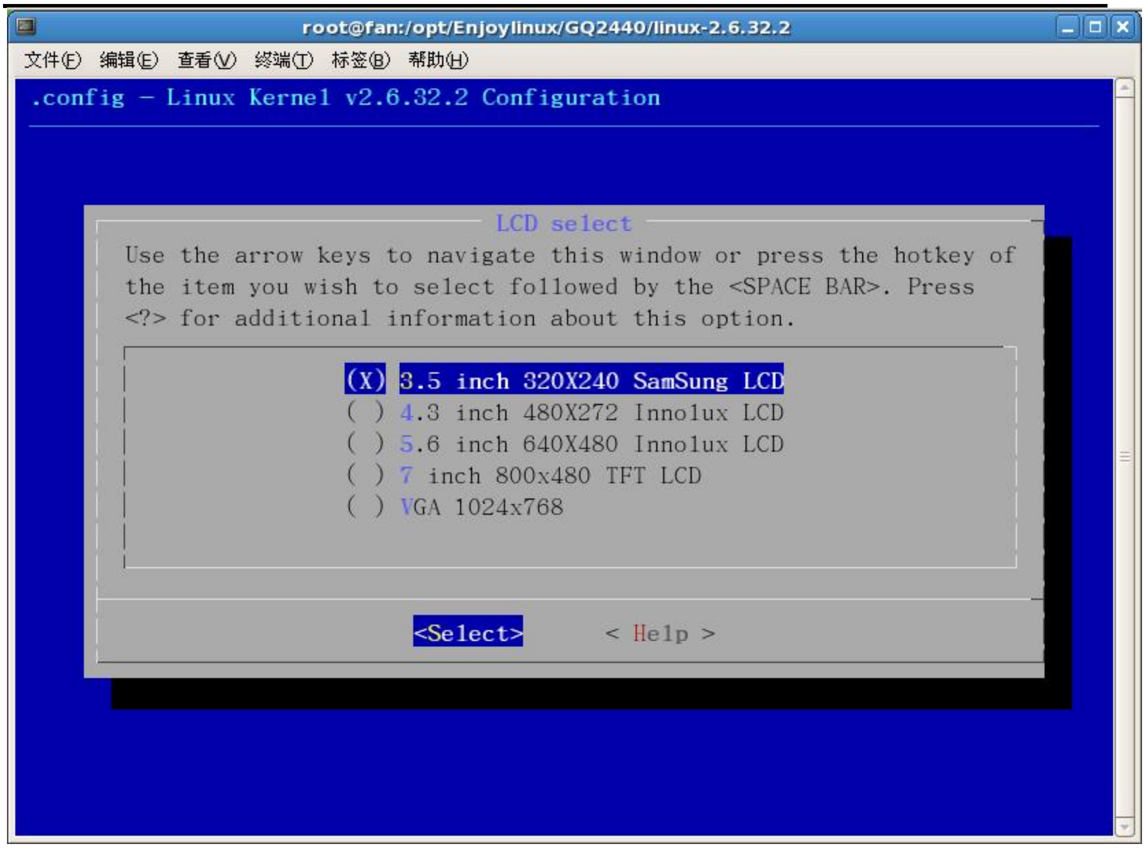
找到如图选项，再按回车进入



出现类似如图界面，并找到如图选项，选中如图 Backlight(背光控制)



再 选 中 LCD select ， 按回车进入， 如图， 可以看到我们加载的默认配置 **config_GQ2440_s35** 在此选择三星 3.5” LCD(3.5 inch 240x320 SamSung LCD)，你还可以根据需要改为其他型号的LCD:

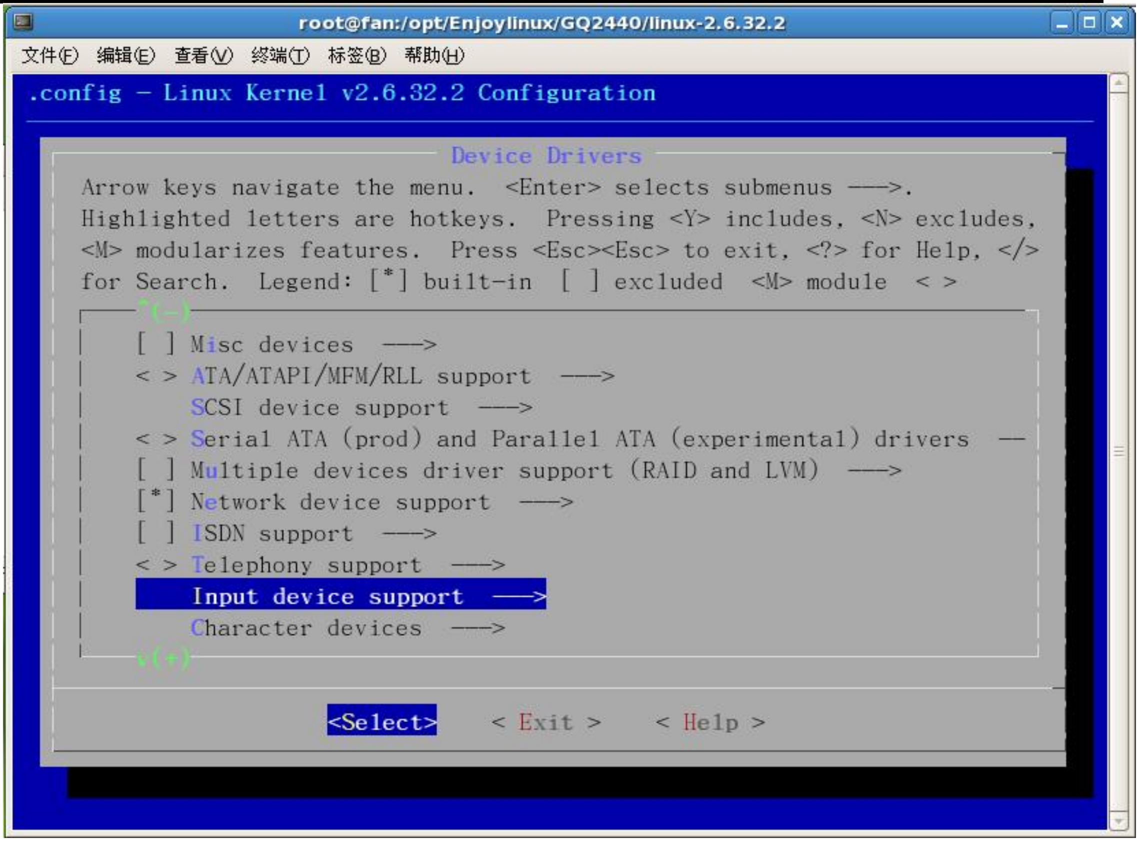


选择完毕，一直按照下方的提示返回到 Device Drivers 配置菜单。

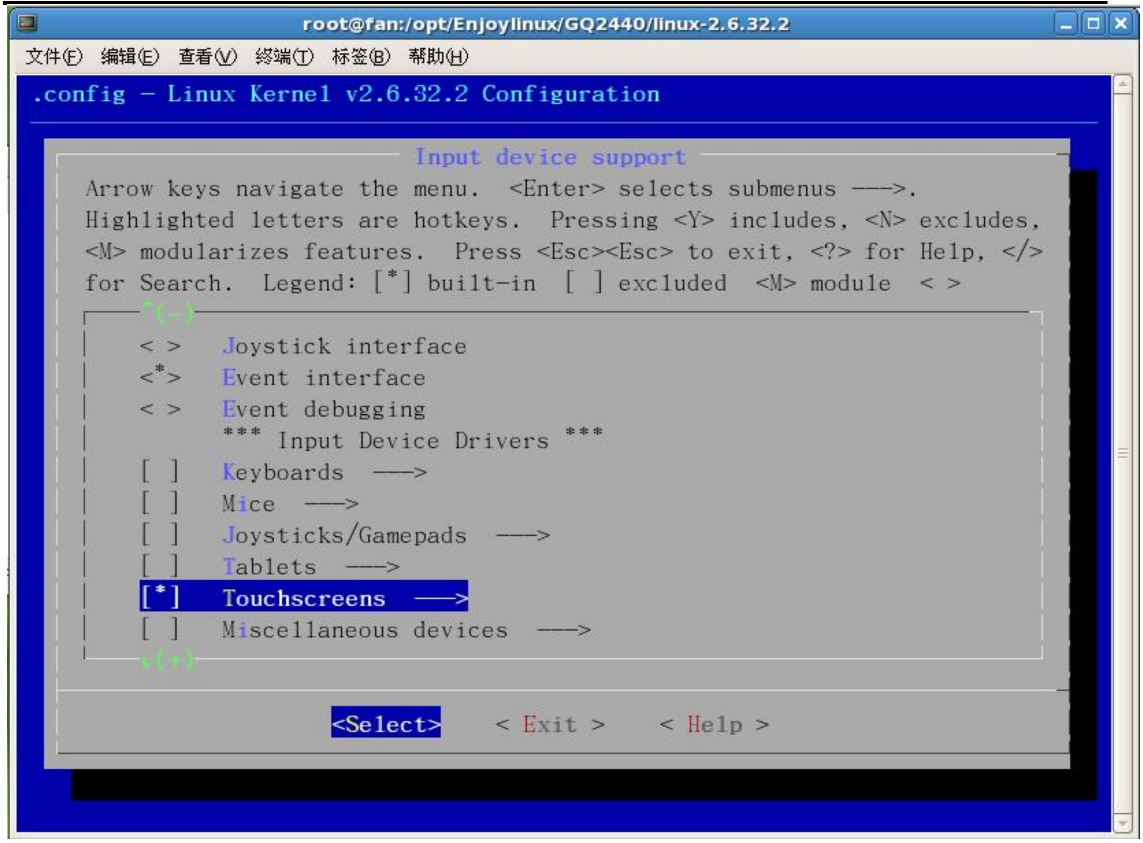
6.3.3 配置触摸屏

注意：如果你选择了VGA1024x768 显示输出模块，是不需要配置此项的。

在 Device Drivers 菜单里面，选择Input device support，按回车进入

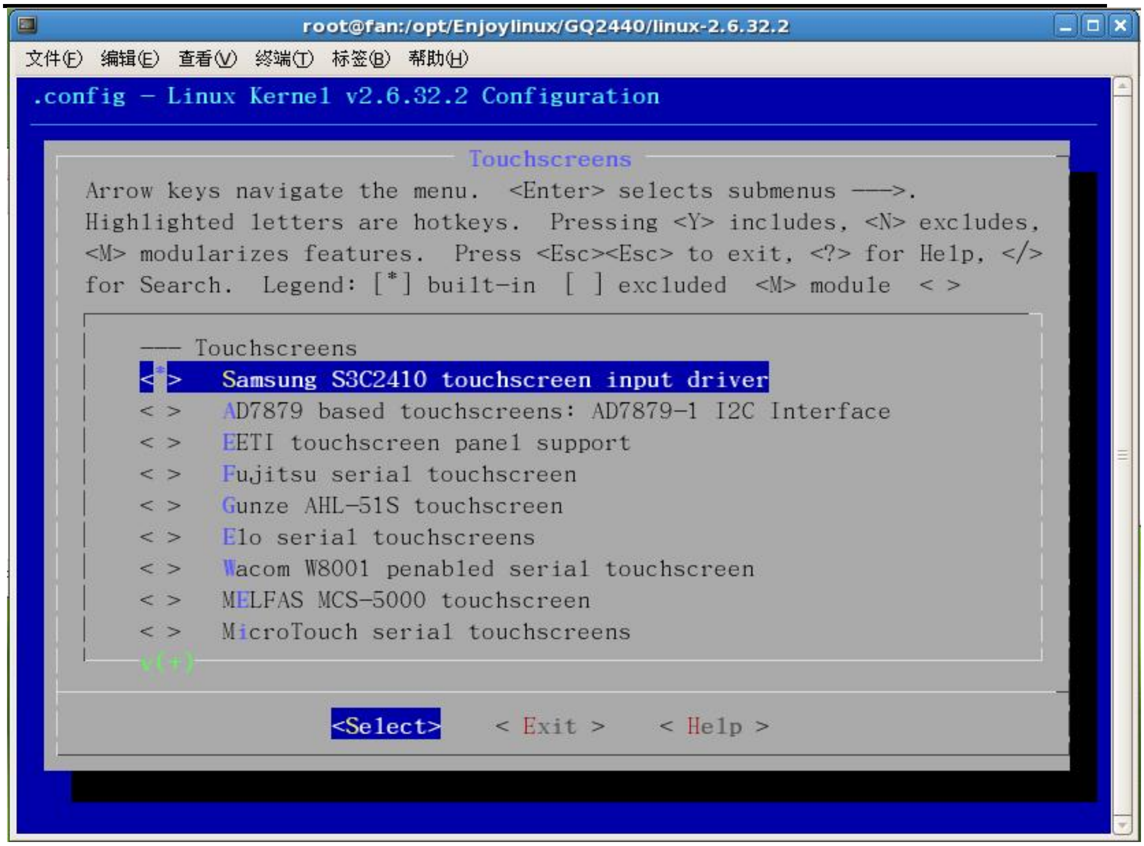


找到并选择 Touchscreens 选项，按回车进入，如图：



root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
.config - Linux Kernel v2.6.32.2 Configuration
Input device support
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < >
^(-)
< > Joystick interface
<*> Event interface
< > Event debugging
*** Input Device Drivers ***
[ ] Keyboards --->
[ ] Mice --->
[ ] Joysticks/Gamepads --->
[ ] Tablets --->
[*] Touchscreens --->
[ ] Miscellaneous devices --->
v(+)
<Select> < Exit > < Help >

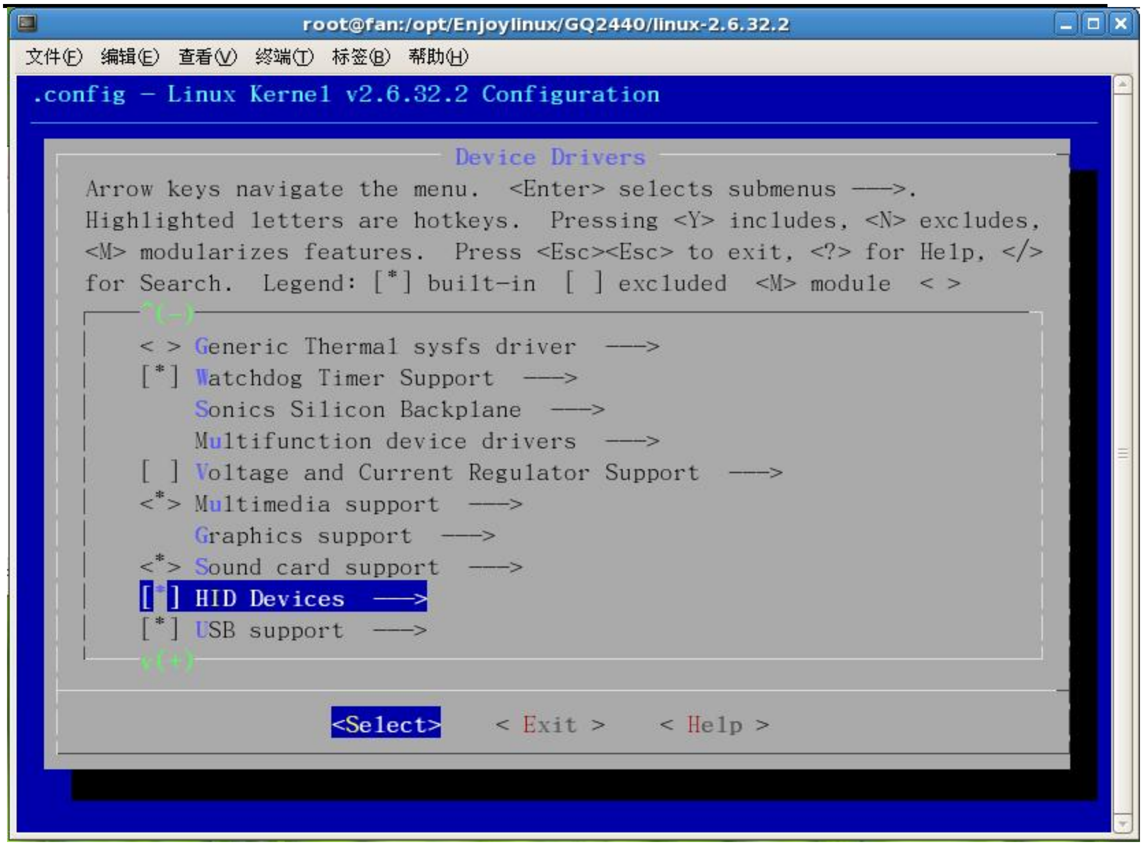
然后如图选择



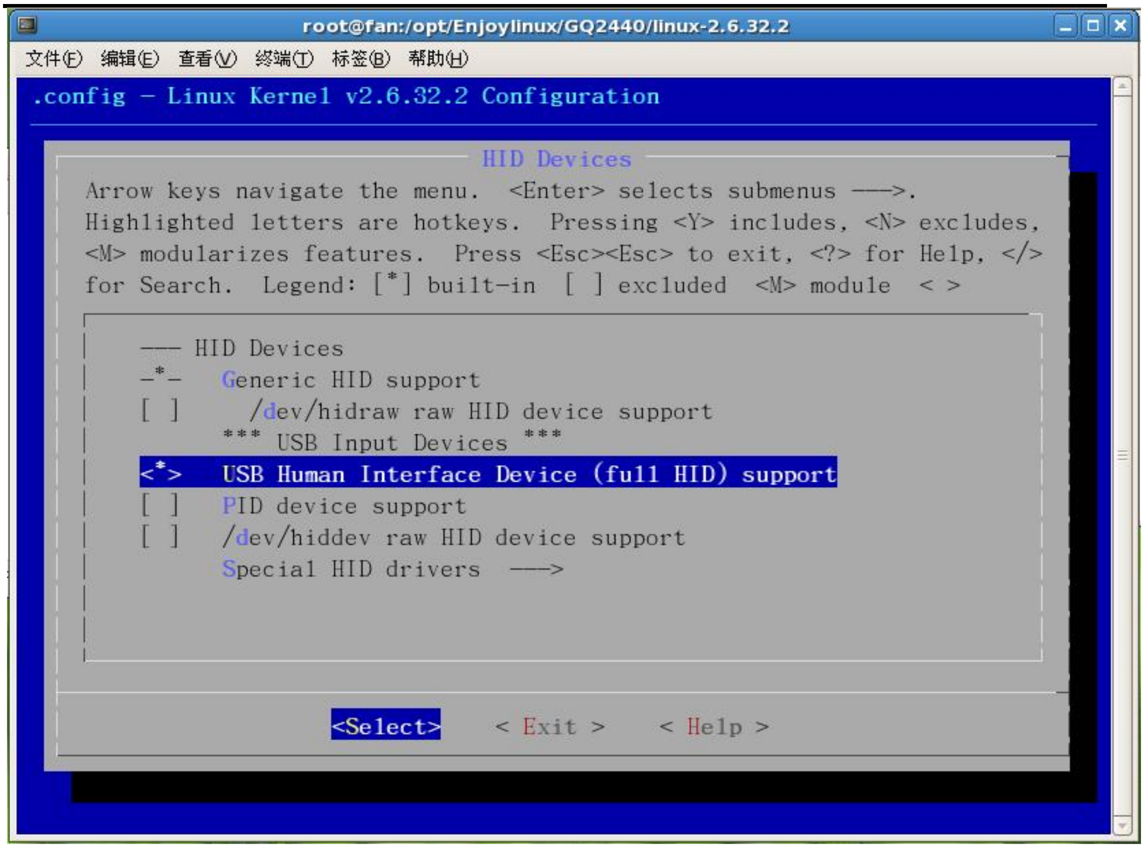
选择完毕，按<Exit>一直返回Device Drivers 菜单。

6.3.4 配置 USB 鼠标和键盘

在 Device Drivers 菜单里面，找到如图选项，并选择进入



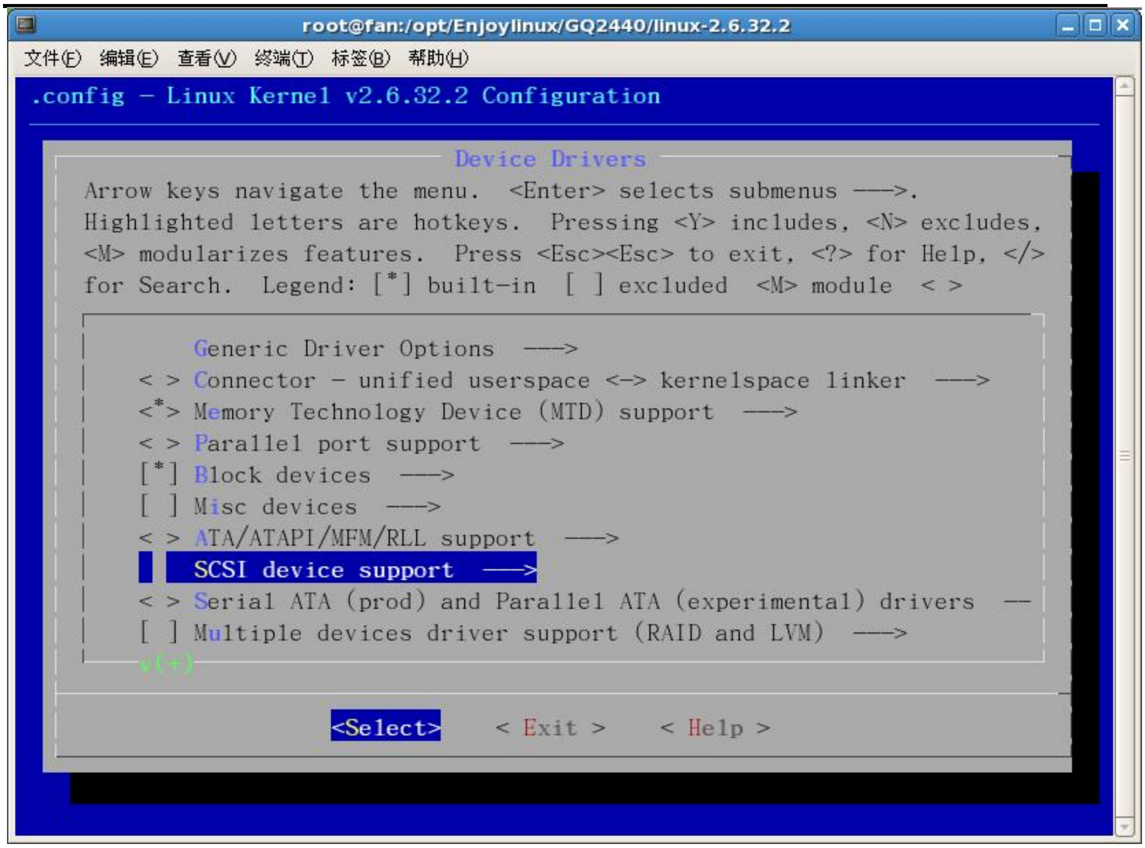
选择如图"*"号所指示的选项



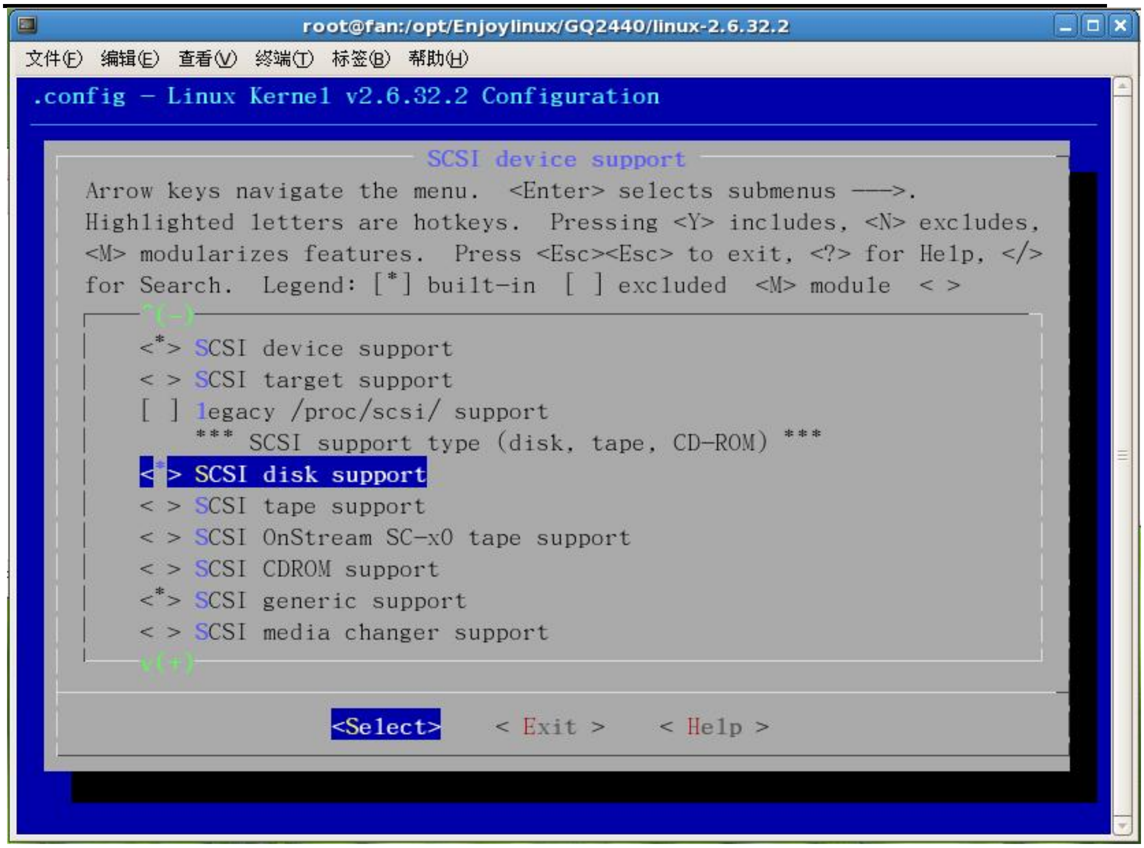
这样就选择配置了 USB 键盘和鼠标，然后选择<Exit>返回 Deice Drivers 菜单。

6.3.5 配置优盘的支持

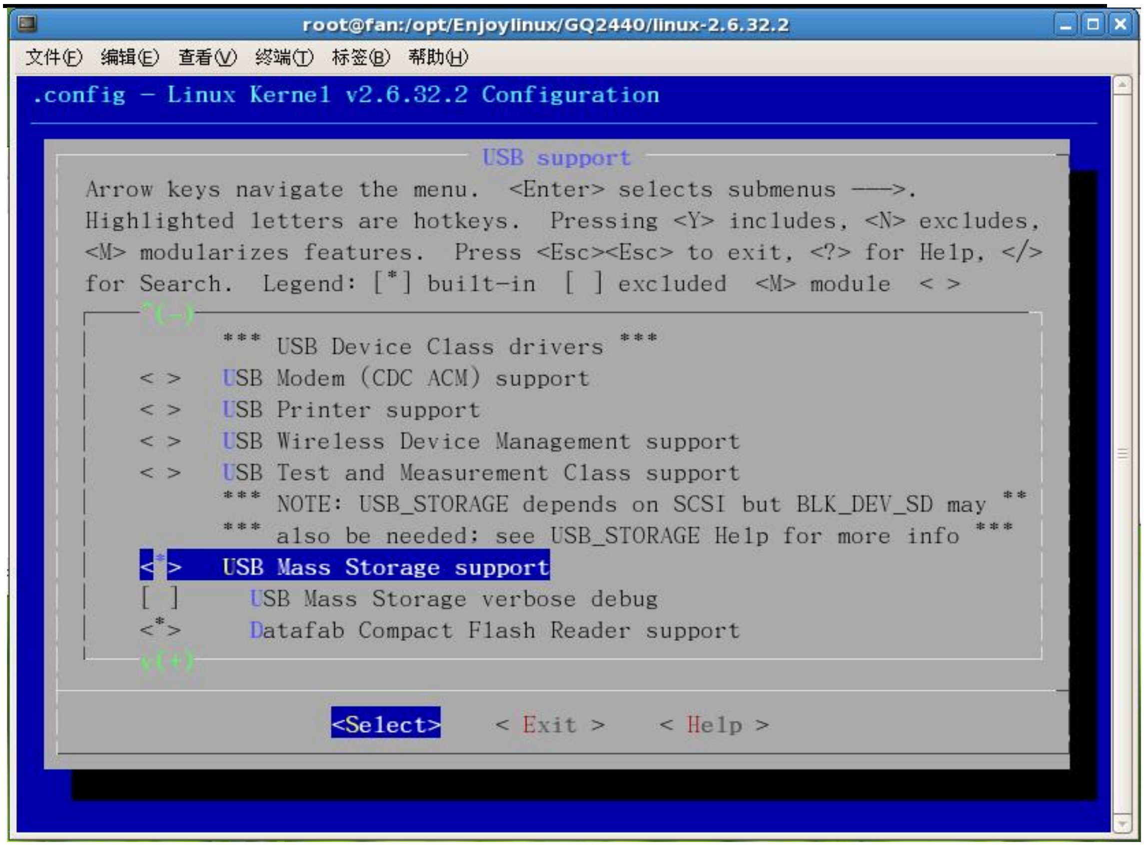
因为优盘用到了 SCSI 命令，所以我们先增加SCSI 支持。在 Device Drivers 菜单里面，选择SCSI device support，按回车进入:



在出现的次菜单中，选择如图:



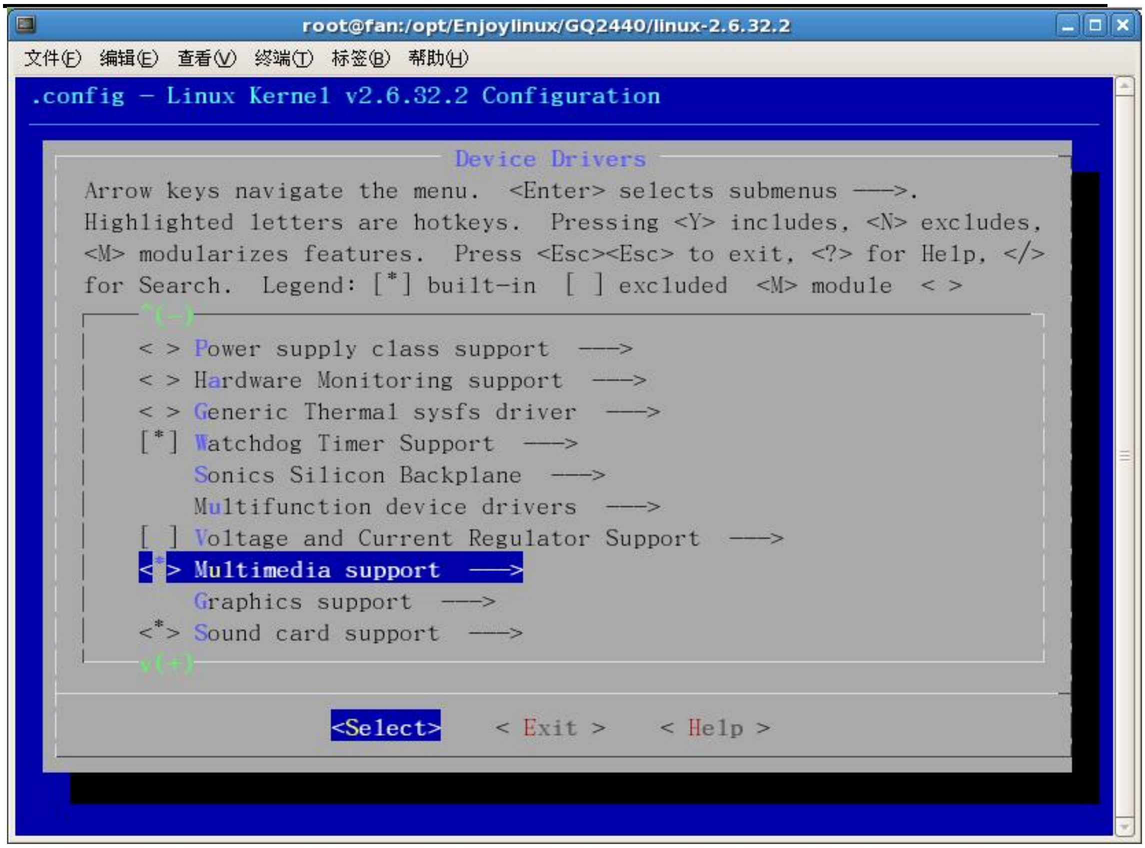
返回 Device Drivers 菜单，再选择 USB support，按回车进入USB support 菜单找到并选中**<*> USB Mass Storage support:**



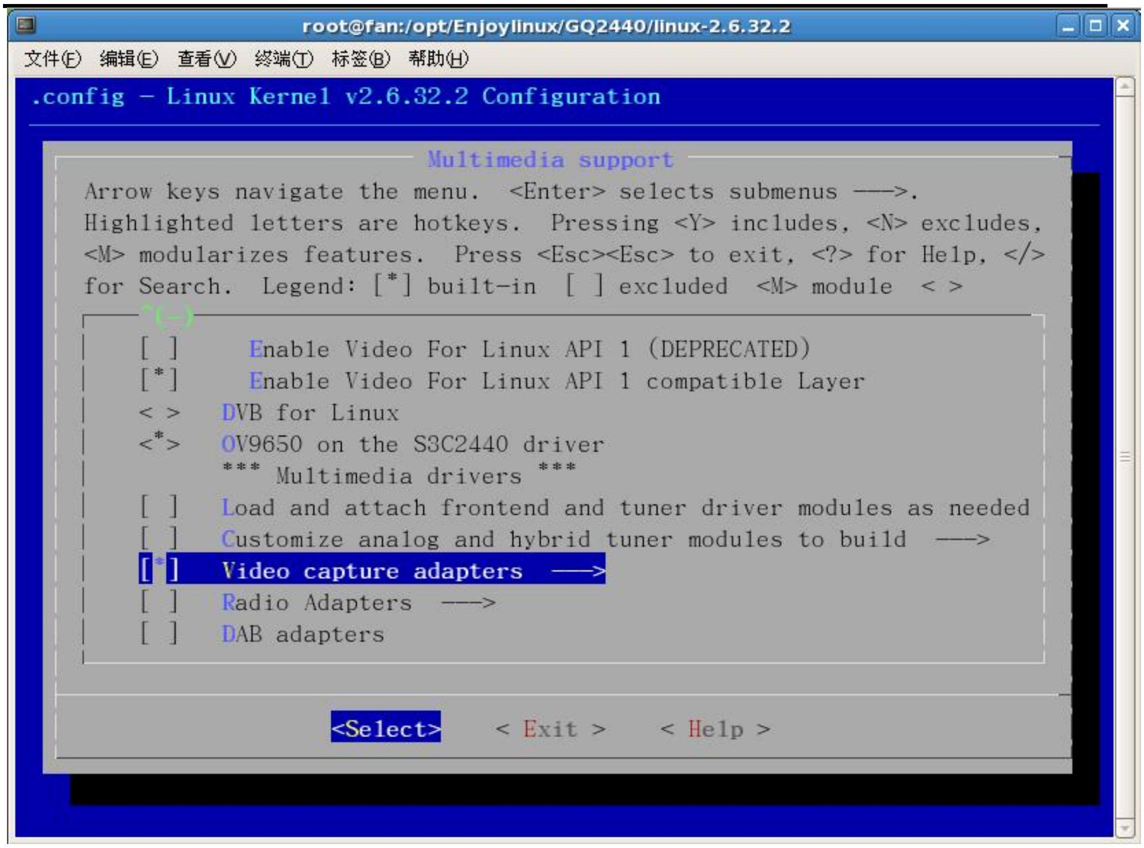
然后选择<Exit>返回Device Drivers 菜单

6.3.6 配置优盘的支持

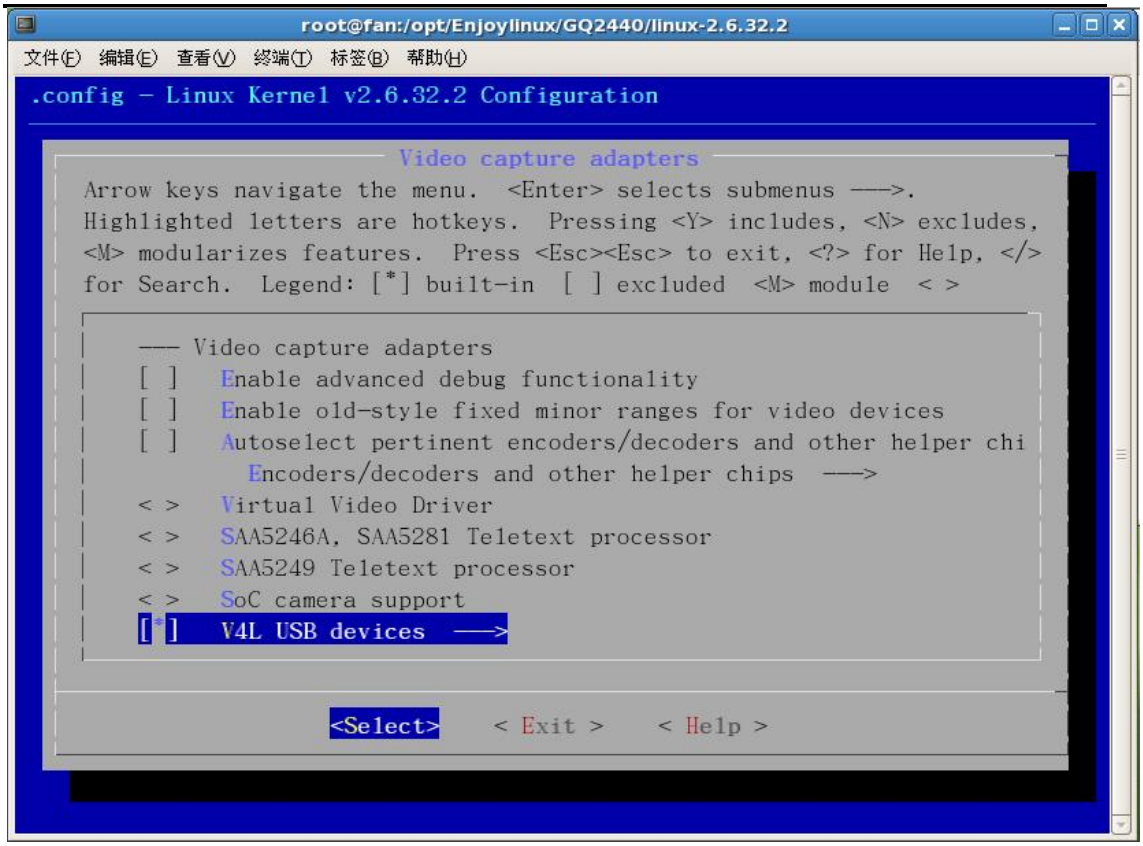
在Device Drivers 菜单里面，选择 Multimedia devices，回车进入：



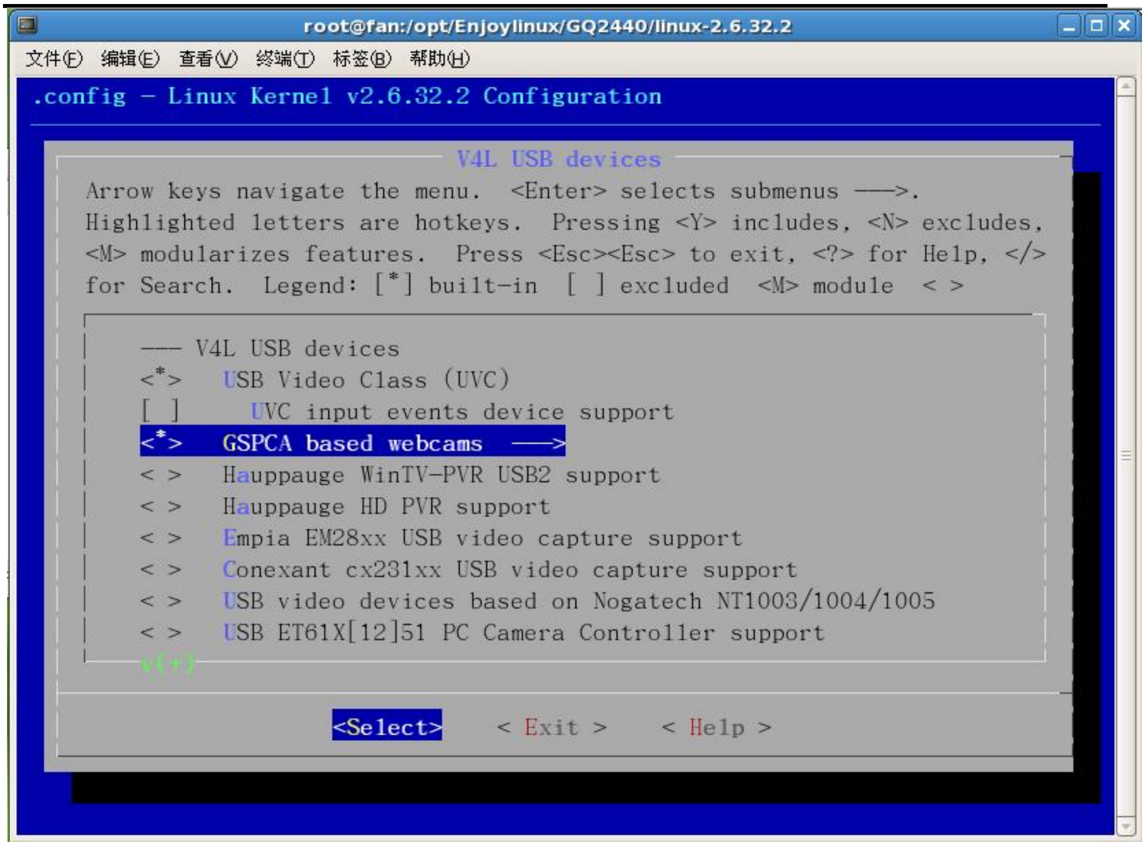
选择如图“*”号选项，并选择Video capture adapters 进入:



出现如图菜单，找到如图选项并进入

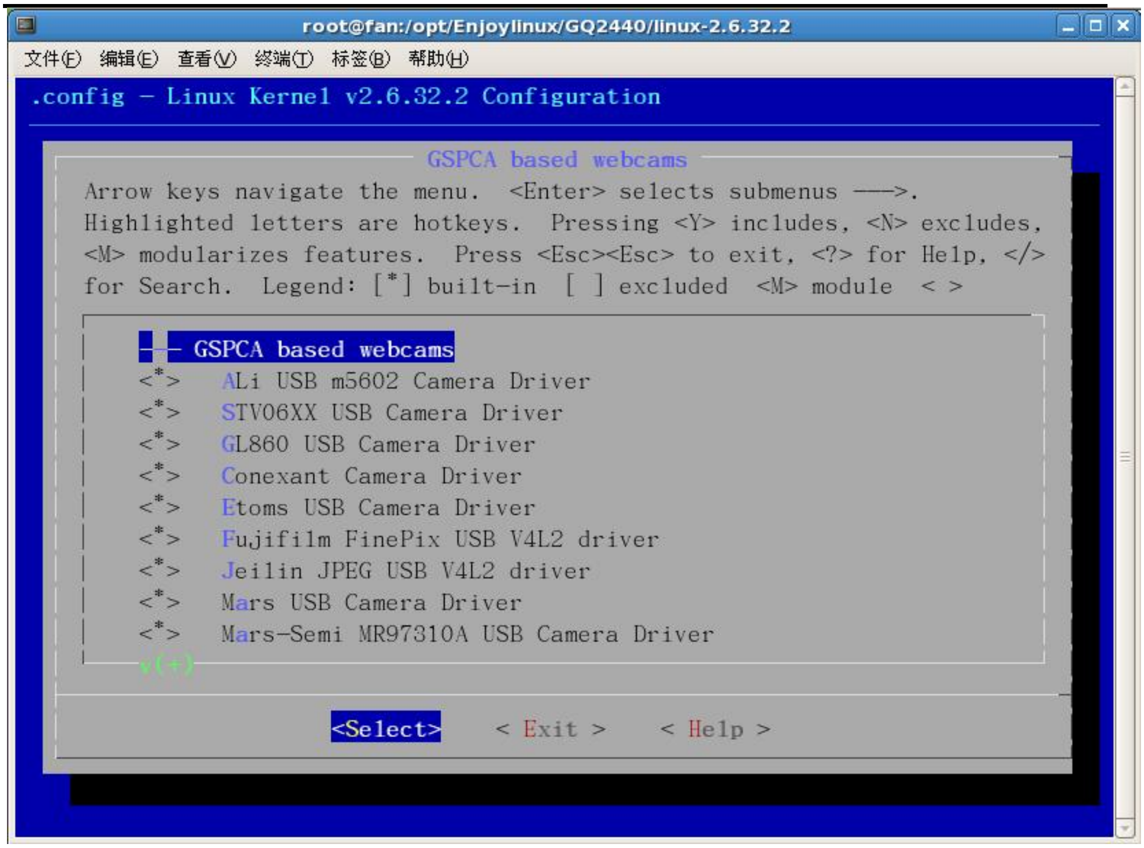


出现如图菜单，选择如图“*”号选项，再选GSPCA based webcams 进入：



GSPCA 是一个法国程序员在业余时间制作的一个万能USB 摄像头驱动程序，在此你可以选择所有类型USB 摄像头的支持，如图

需要注意的是：虽然这里选择了众多型号的摄像头驱动，但每个型号的Video 输出格式并不完全相同，这需要在高层应用中根据实际情况分别做处理，才能正常使用这些驱动。

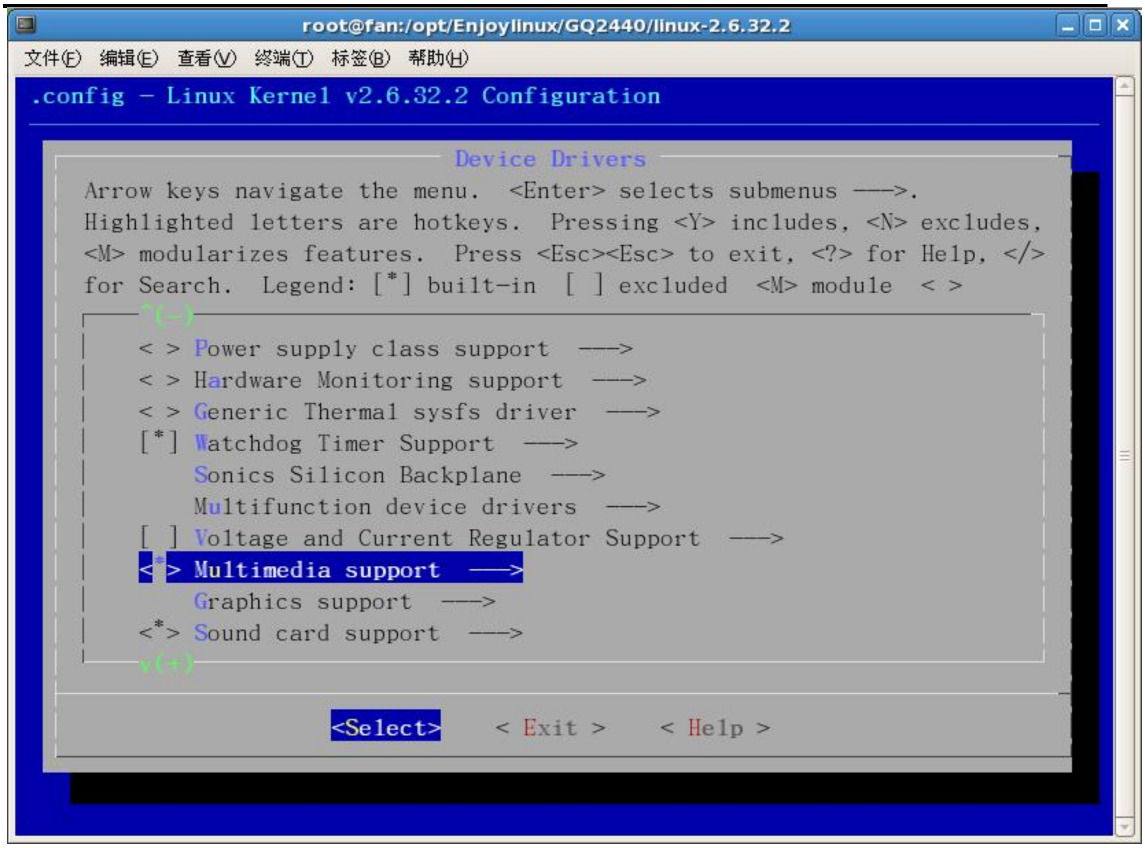


一直选择<Exit>返回Device Drivers 菜单，再选择<Exit>返回到主菜单。

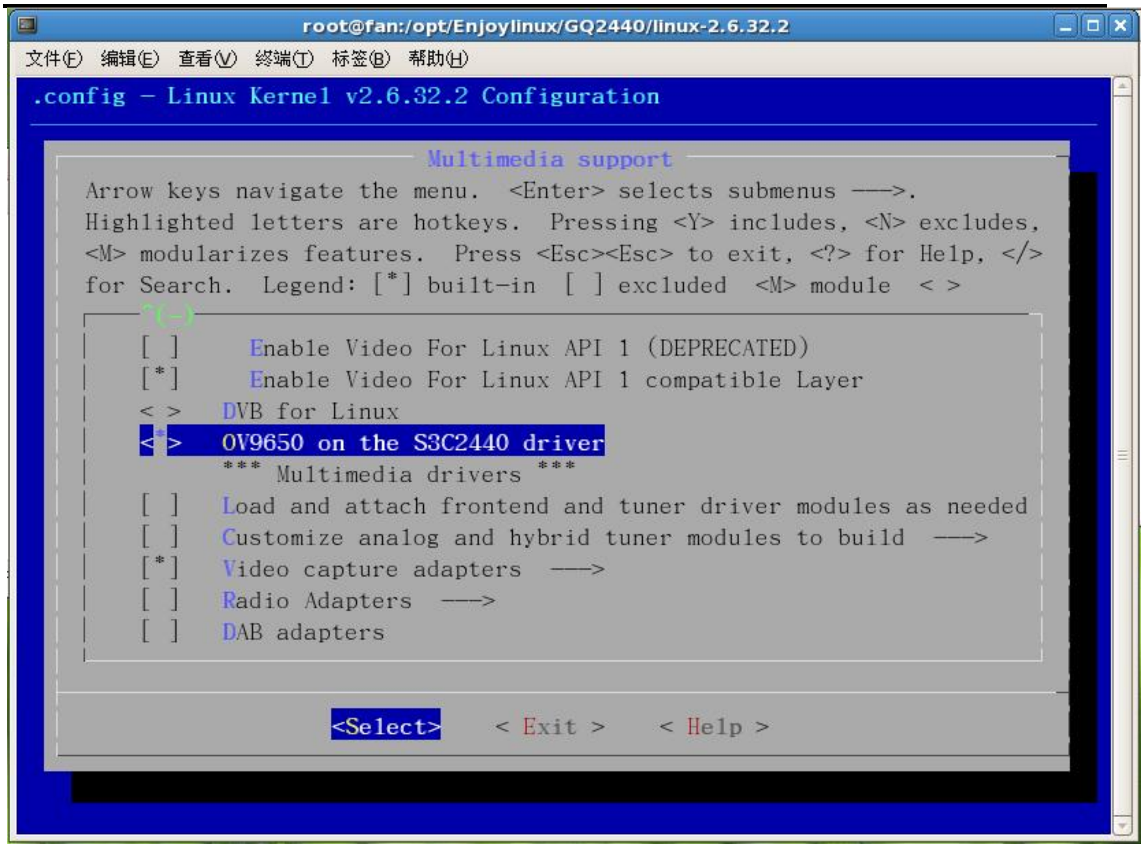
6.3.7 配置 CMOS 摄像头驱动

本开发板配用的 CMOS 摄像头模块CAM130，其内部使用的OV9650 芯片，因此我们需要为此配置驱动程序，如下步骤：

在 Device Drivers 菜单里面，选择 Multimedia devices，回车进入：



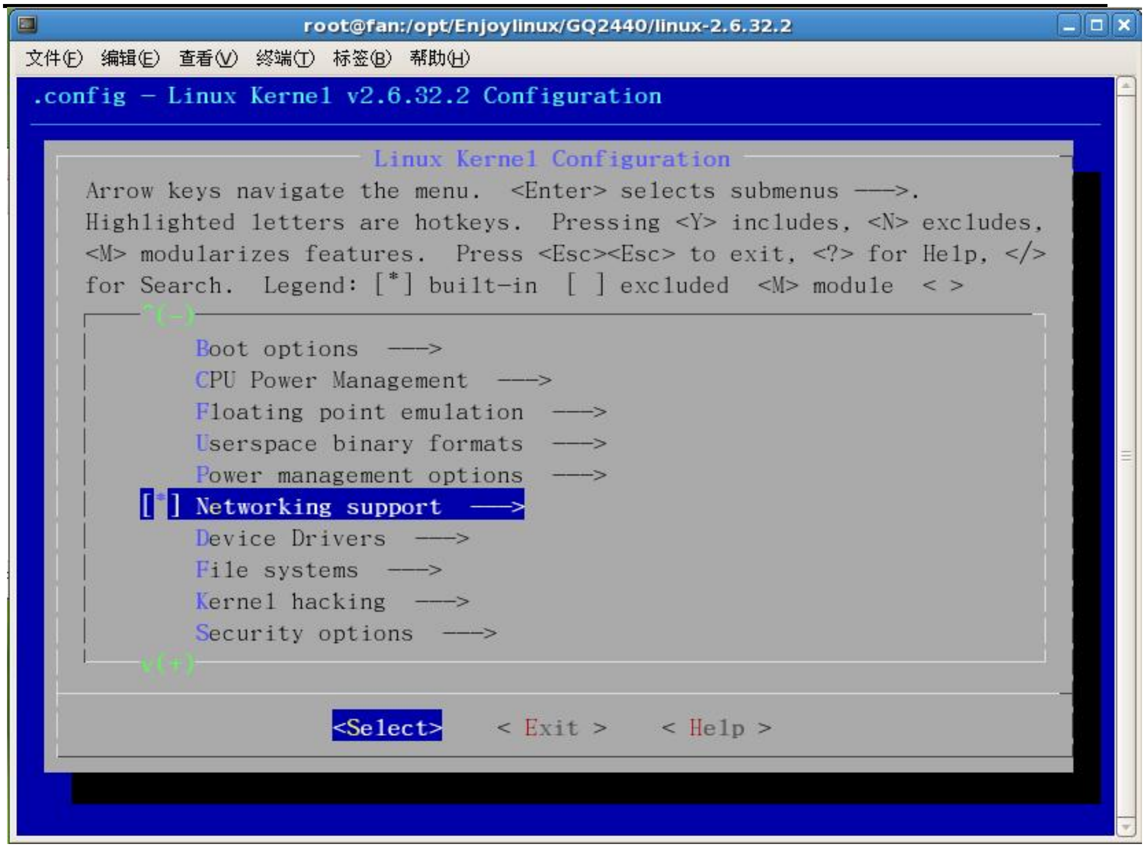
选择如图“*”号选项，并选择Video capture adapters 进入，找到OV9650 芯片驱动并选中它，如图：



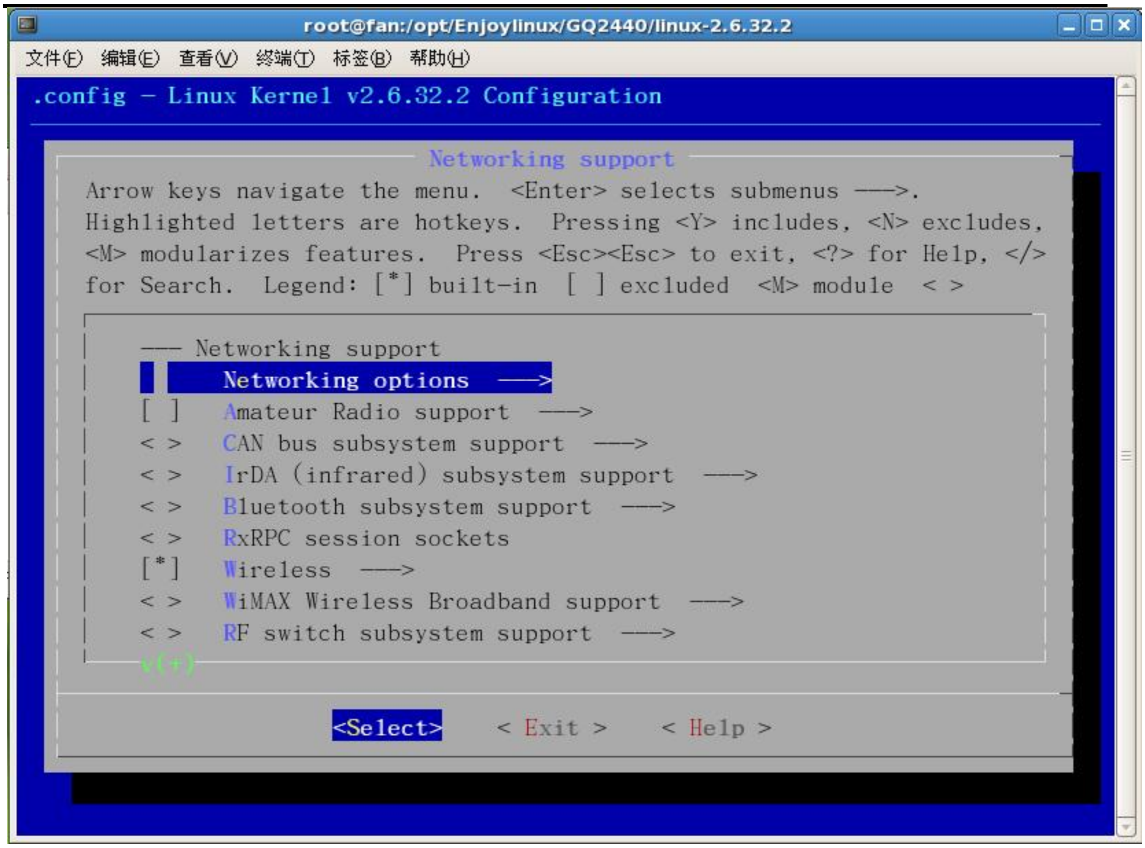
注意： 我们为CAM130 模块设计的驱动程序，既不属于VL4 体系，也不属于V4L2体系，它就是一个简单的字符设备，这样做是为了方便移植。

6.3.8 配置网卡驱动

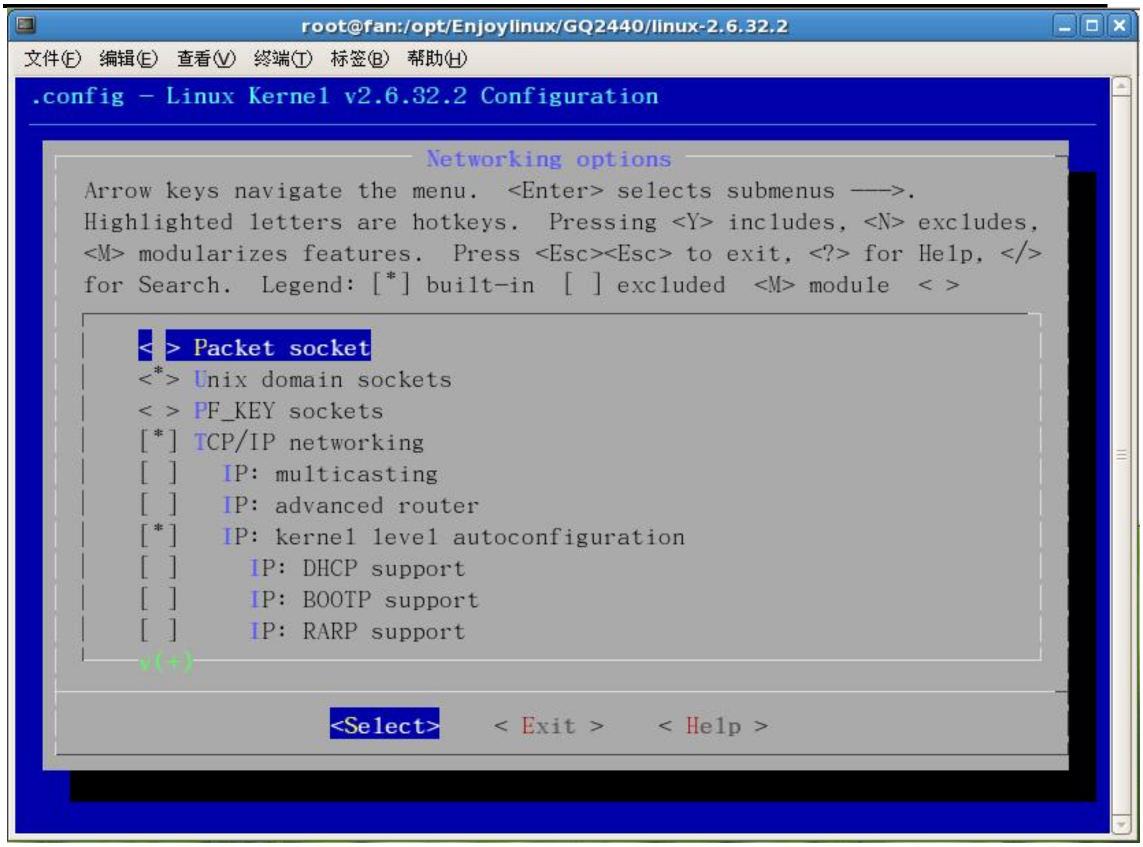
要配置网卡驱动，首先要配置网络协议支持。在主菜单中，选择 Netwoking support，回车进入：



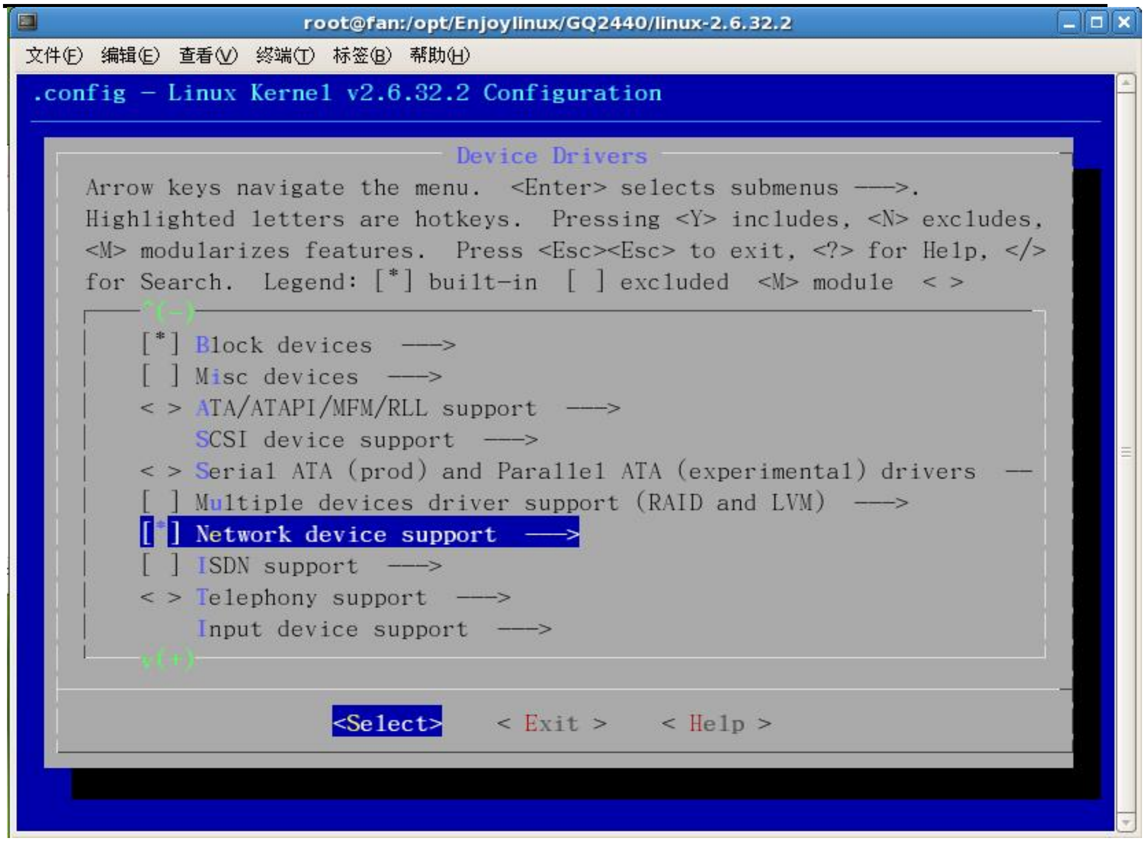
出现如图子菜单，如图选择 Networking options 并进入：



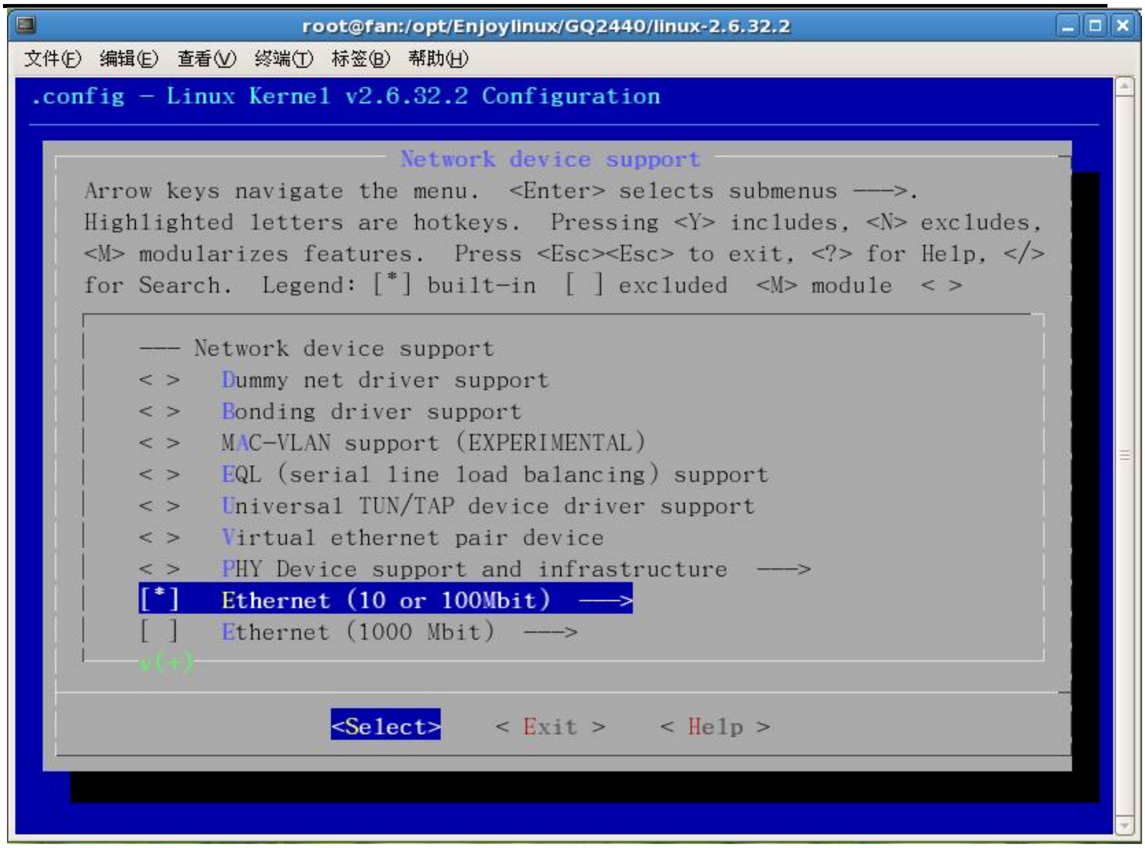
一般我们选择 TCP/IP 协议就够了，但推荐使用我们缺省配置的几个选项，如图



选择完毕，一直退回到主菜单，并选择进入 Device Drivers 菜单。找到 Network device support，选择进入：



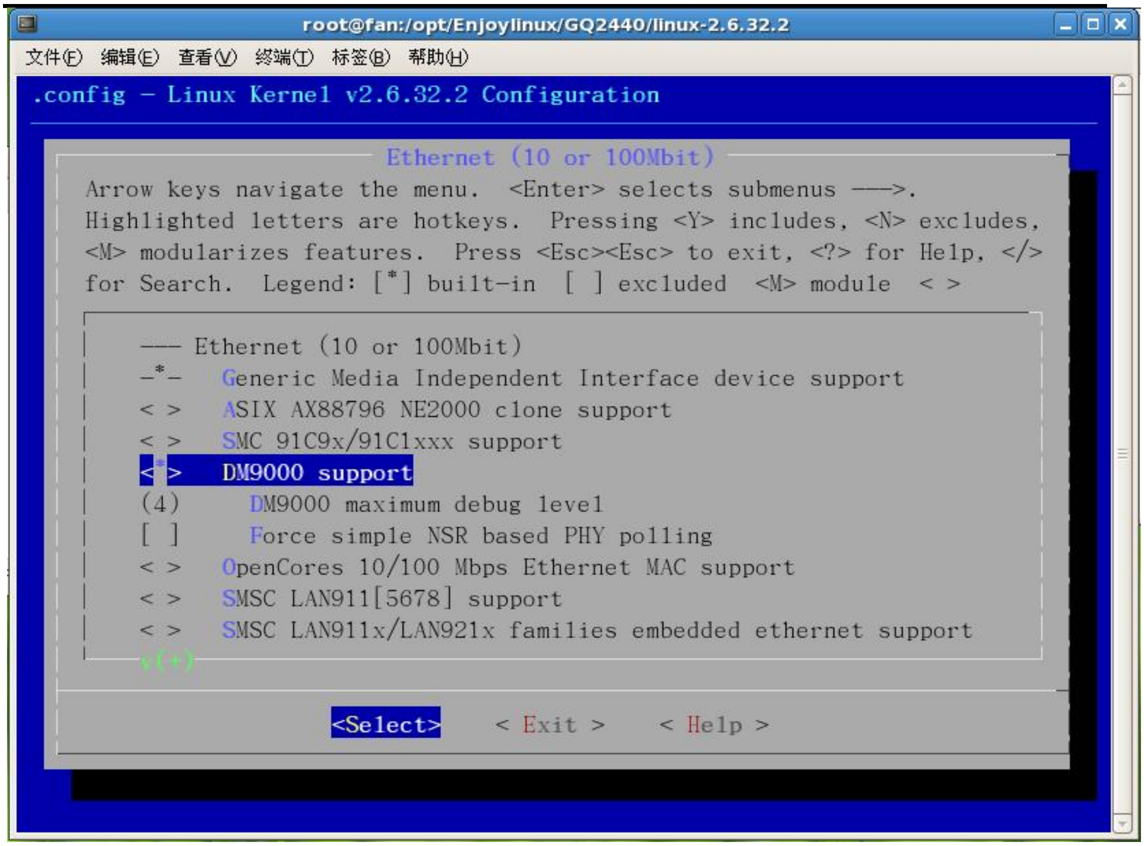
找到并进入 Ethernet (10 or 100Mbit) 选项



选中:

<*> Generic Media Independent Interface device support

<*> DM9000 support

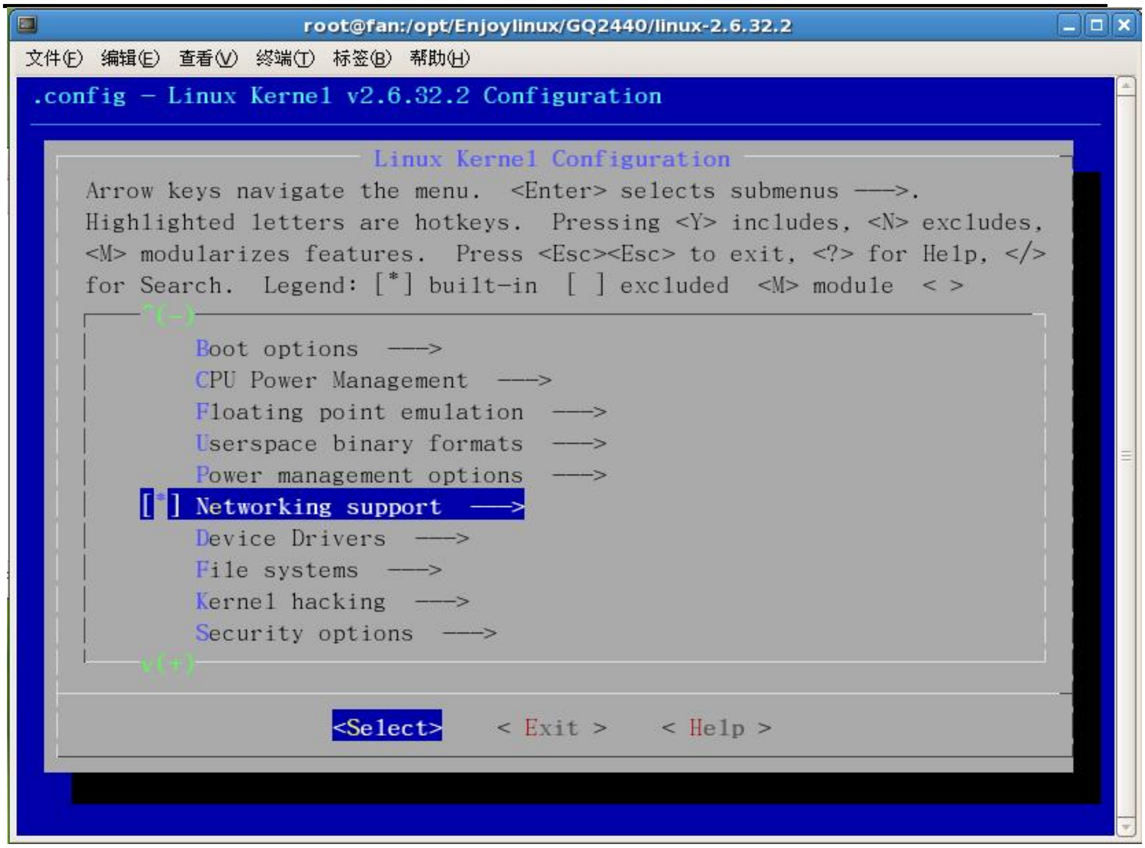


选择<Exit>一直返回到Device Drivers 菜单。

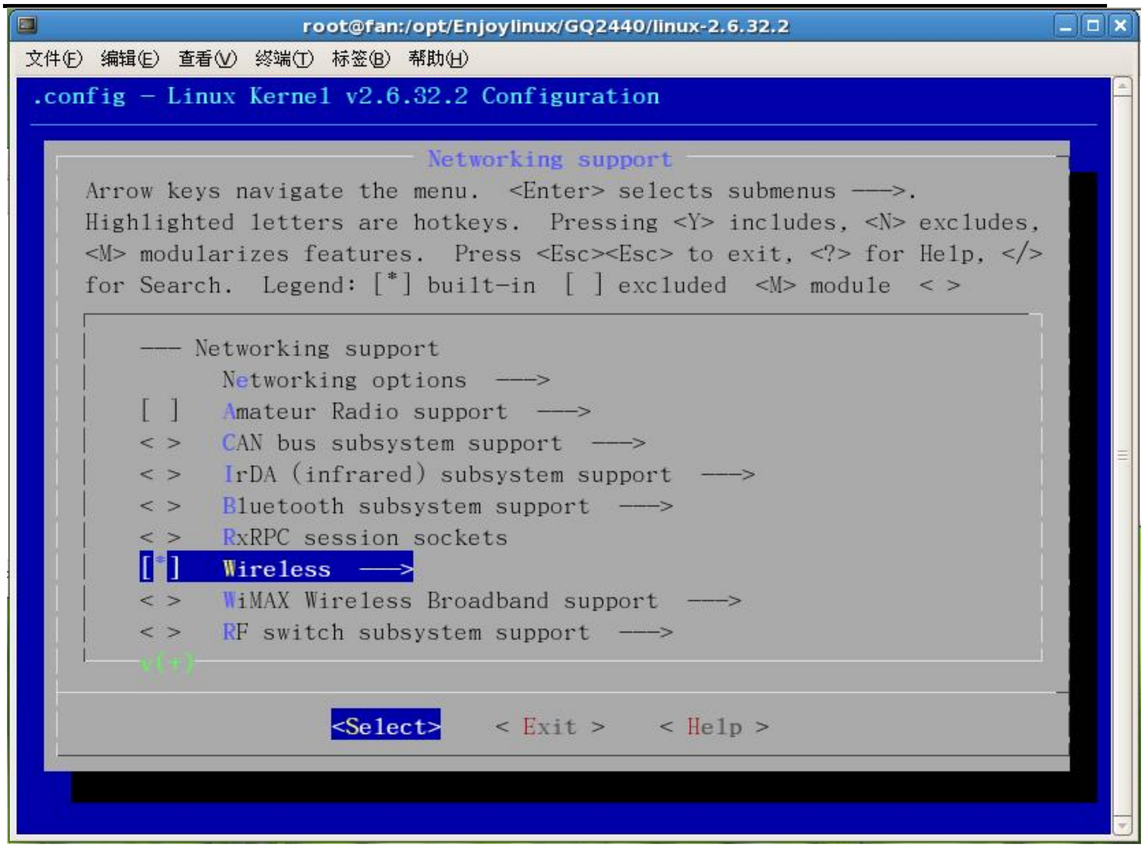
6.3.9 配置 USB 无线网卡驱动

本开发板采用 Linux-2.6.32.2 内核，它已经包含了多种型号的USB 无线网卡驱动，在
我们提供的缺省配置中，也已经包含了大部分常见的网卡型号，如TP-Link 系列，VIA 系列等，下面是它的驱动配置说明。

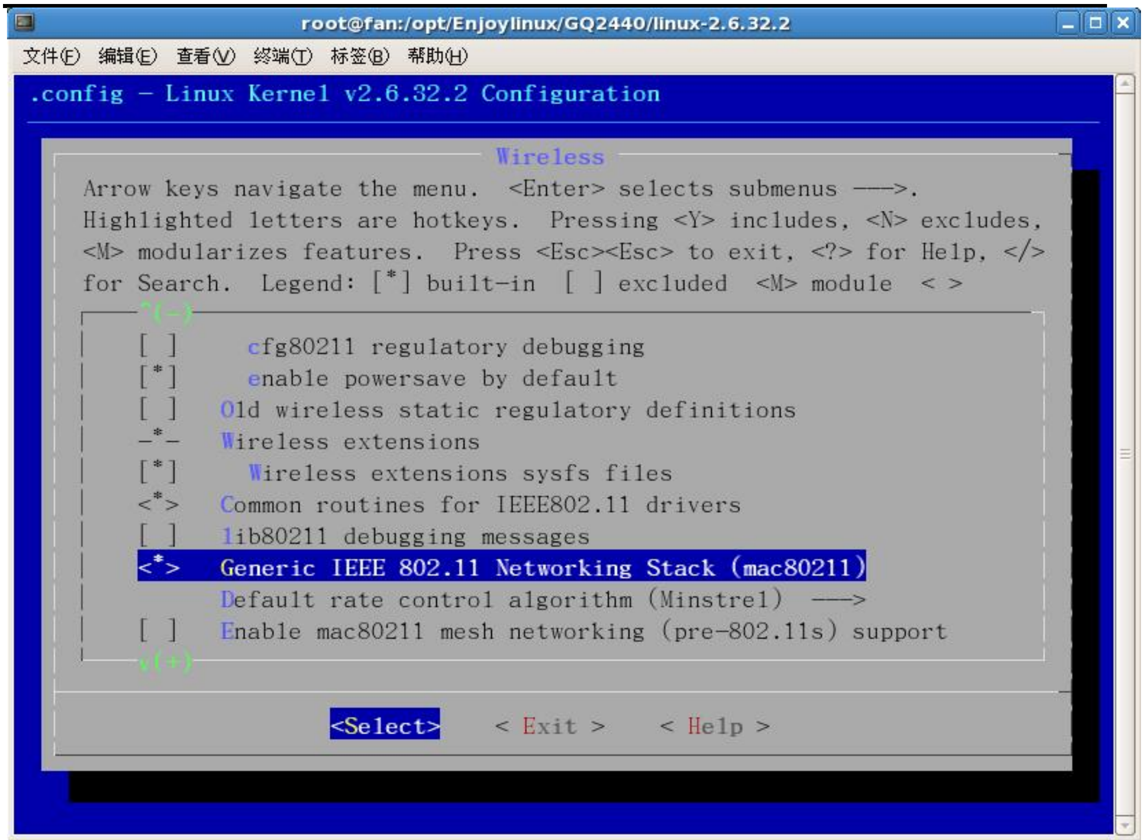
在主菜单中，选择 Netwoking support，回车进入：



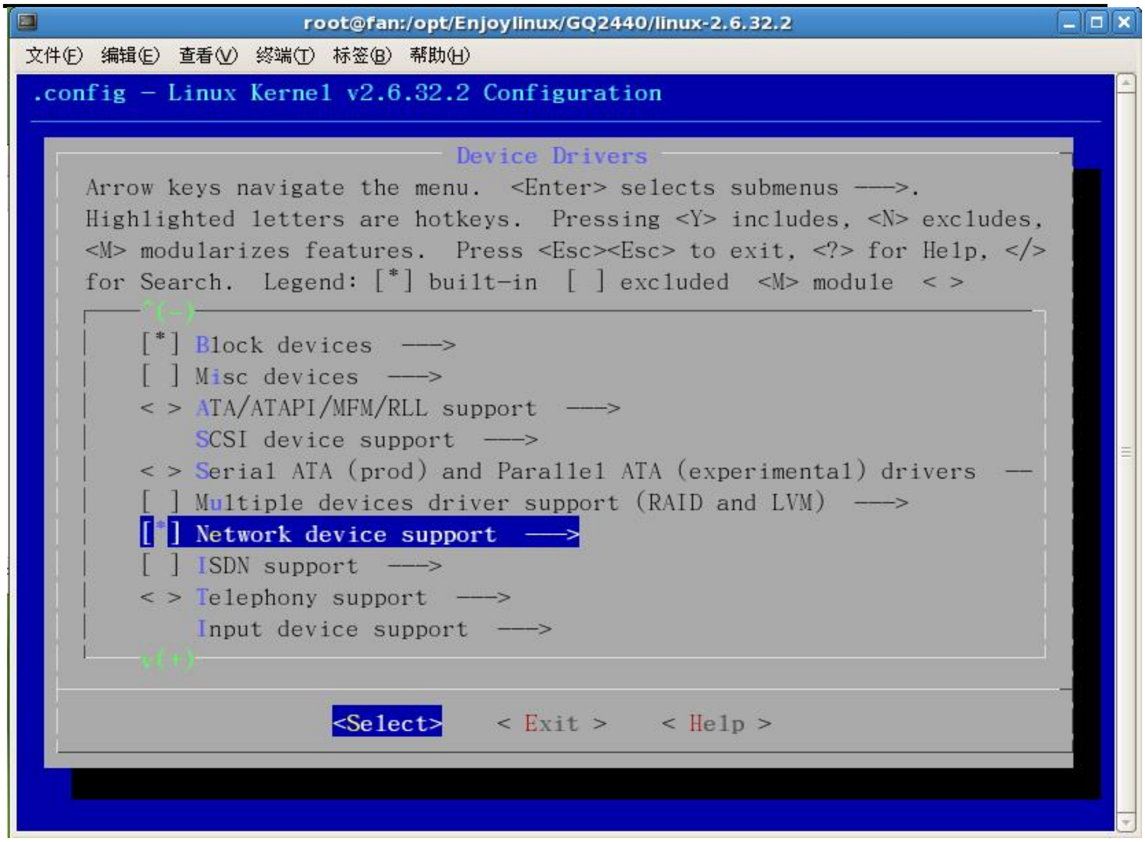
出现如图子菜单，如图选择 Wireless 并进入开始配置无线网络协议


root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
.config - Linux Kernel v2.6.32.2 Configuration
Networking support
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < >
--- Networking support
Networking options --->
[ ] Amateur Radio support --->
< > CAN bus subsystem support --->
< > IrDA (infrared) subsystem support --->
< > Bluetooth subsystem support --->
< > RxRPC session sockets
[*] Wireless --->
< > WiMAX Wireless Broadband support --->
< > RF switch subsystem support --->
v(+)
<Select> < Exit > < Help >

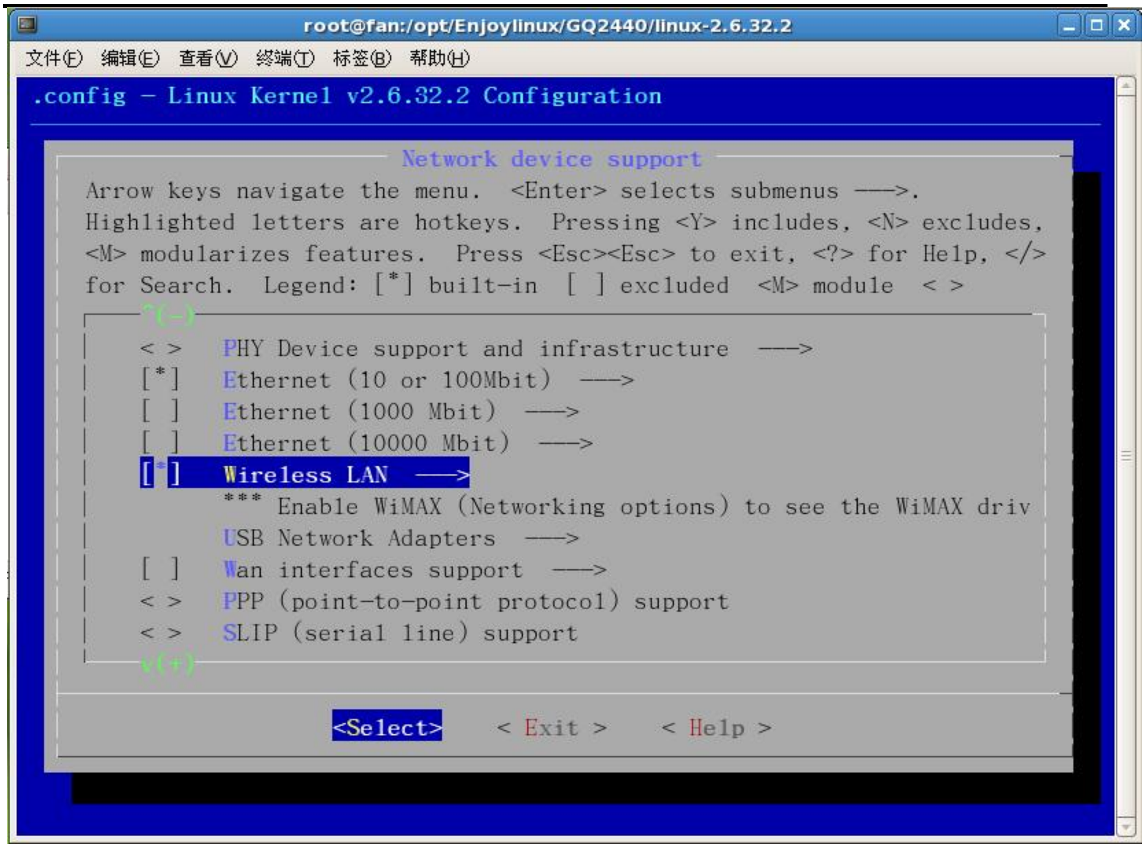
选择如图*各项配置:



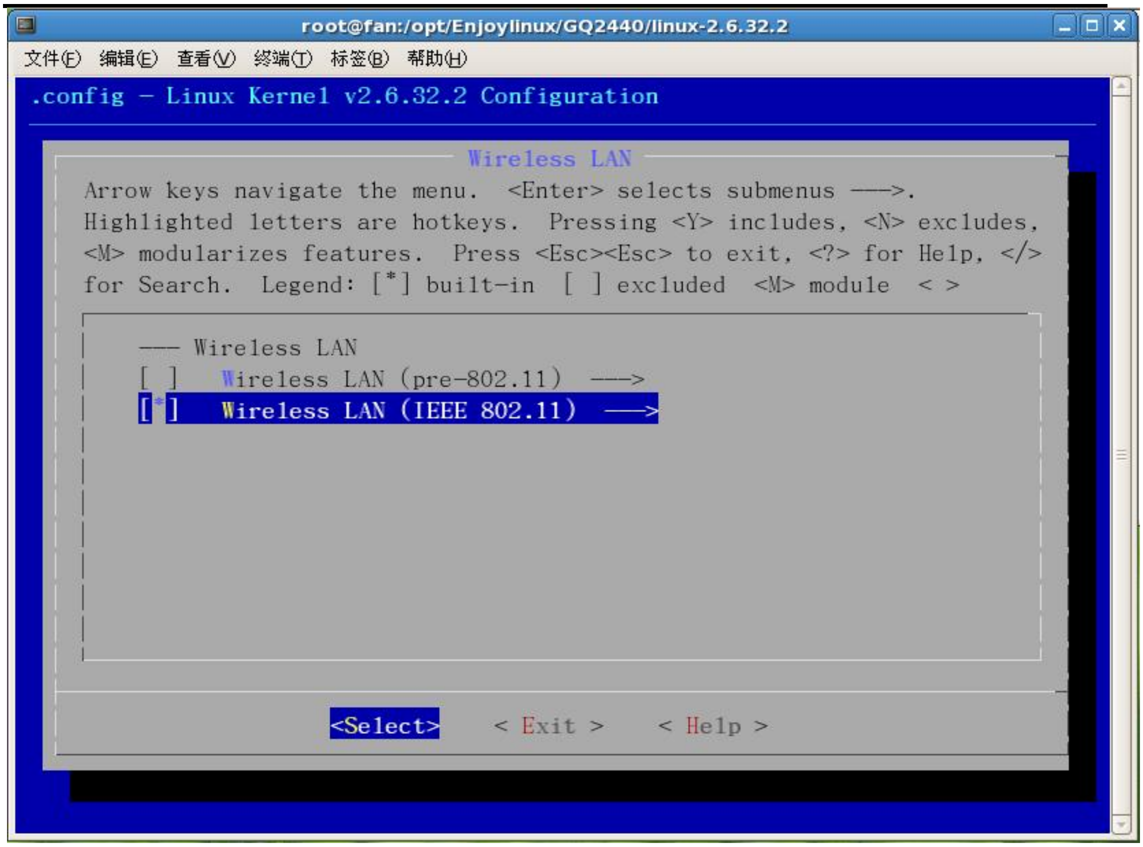
退回到内核配置主菜单，选择 Device Drivers 并进入，开始配置无线网卡驱动，如图



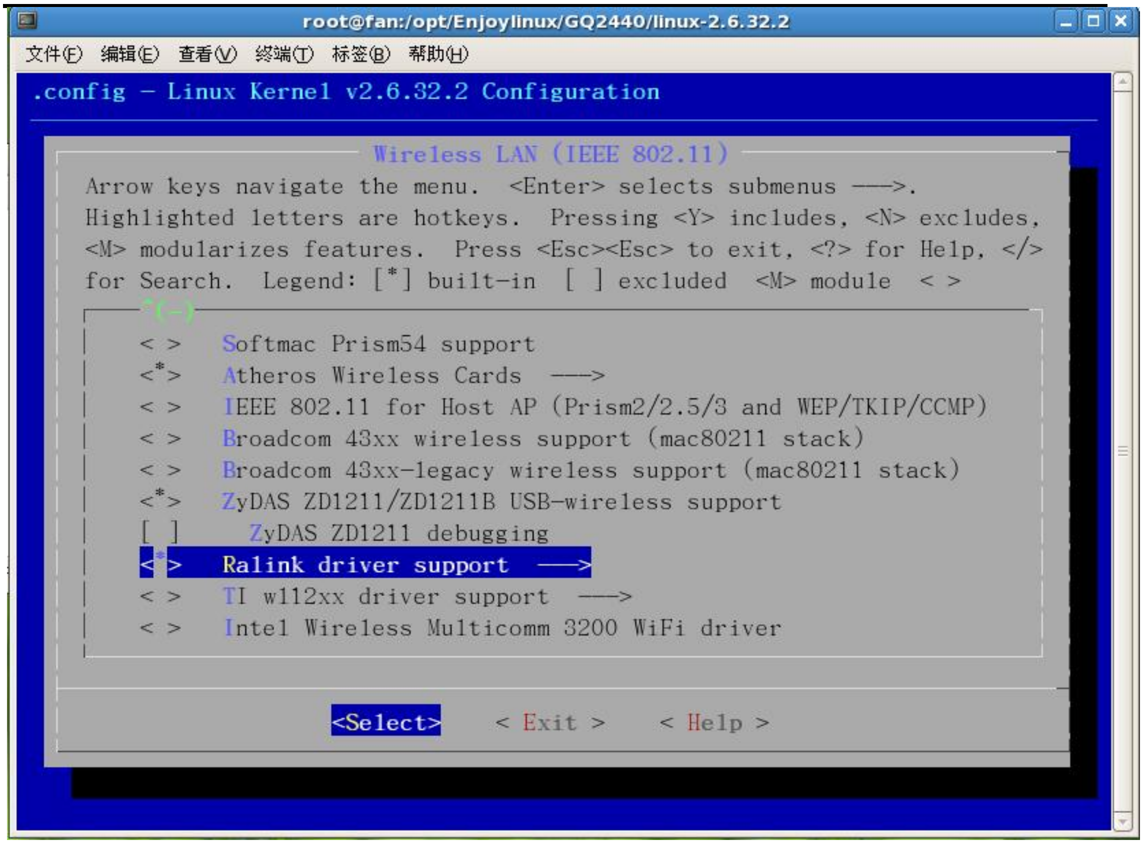
进入网络设备子菜单，找到如图无线网络设备子项，并进入

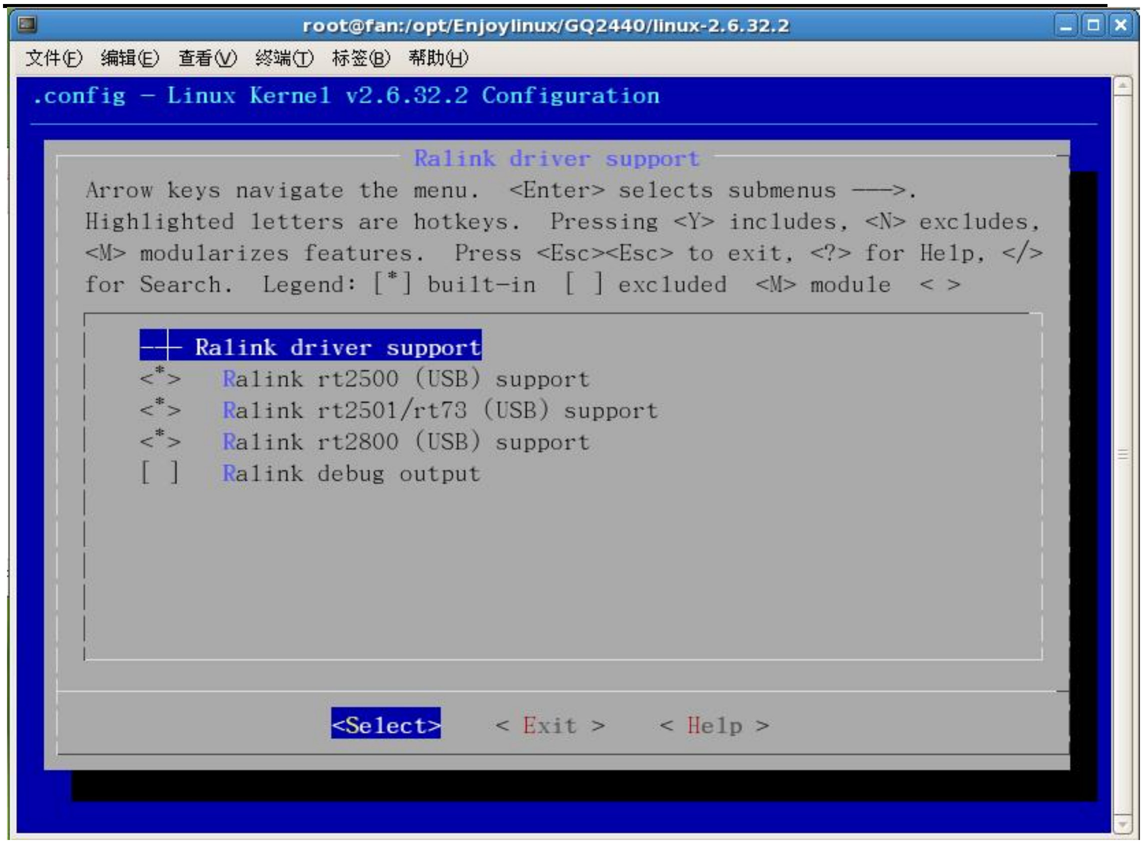


再选择 Wireless LAN(IEEE 802.11)子项，并进入



可以看到已经配置了以芯片厂商为分类方式的常见各种 USB 无需网卡类新，如图为 Ralink 公司芯片方案的USB 无线网卡驱动支持:

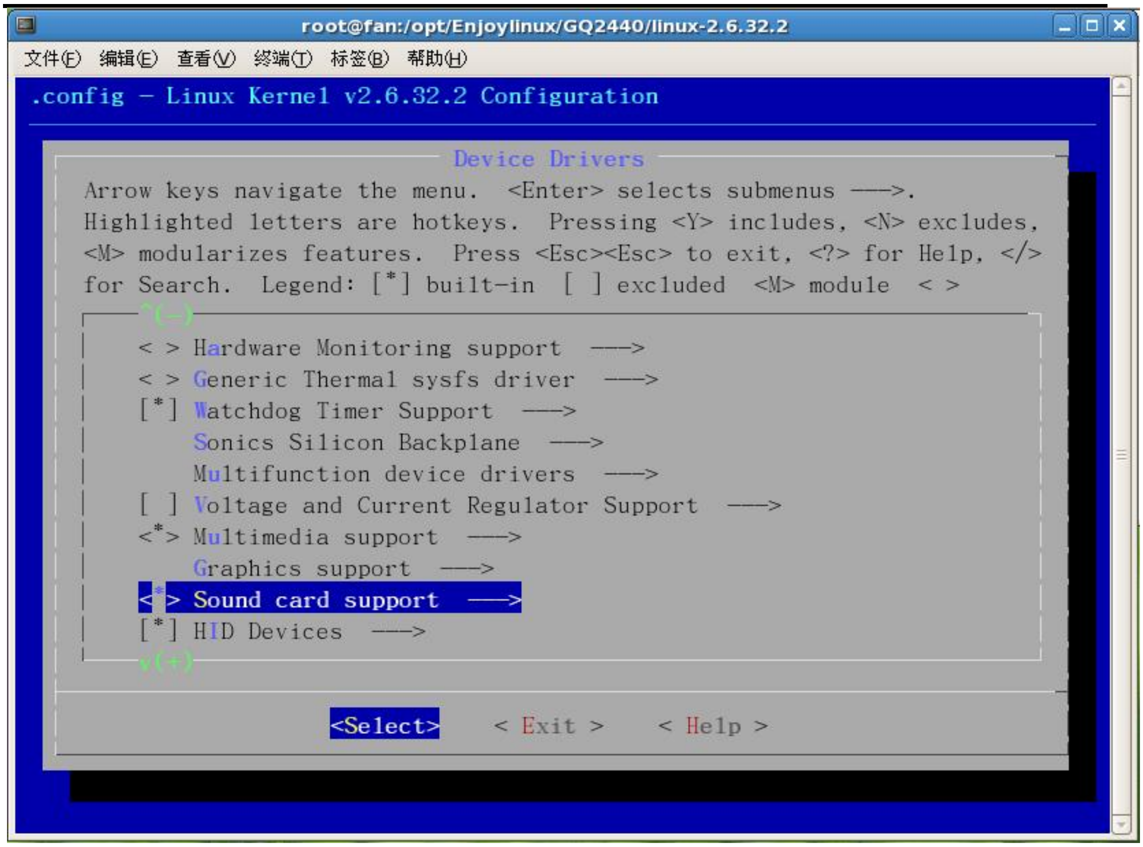

root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
.config - Linux Kernel v2.6.32.2 Configuration
Wireless LAN (IEEE 802.11)
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
< > Softmac Prism54 support
<*> Atheros Wireless Cards --->
< > IEEE 802.11 for Host AP (Prism2/2.5/3 and WEP/TKIP/CCMP)
< > Broadcom 43xx wireless support (mac80211 stack)
< > Broadcom 43xx-legacy wireless support (mac80211 stack)
<*> ZyDAS ZD1211/ZD1211B USB-wireless support
[ ] ZyDAS ZD1211 debugging
<*> Ralink driver support --->
< > TI wl12xx driver support --->
< > Intel Wireless Multicomm 3200 WiFi driver
<Select> < Exit > < Help >



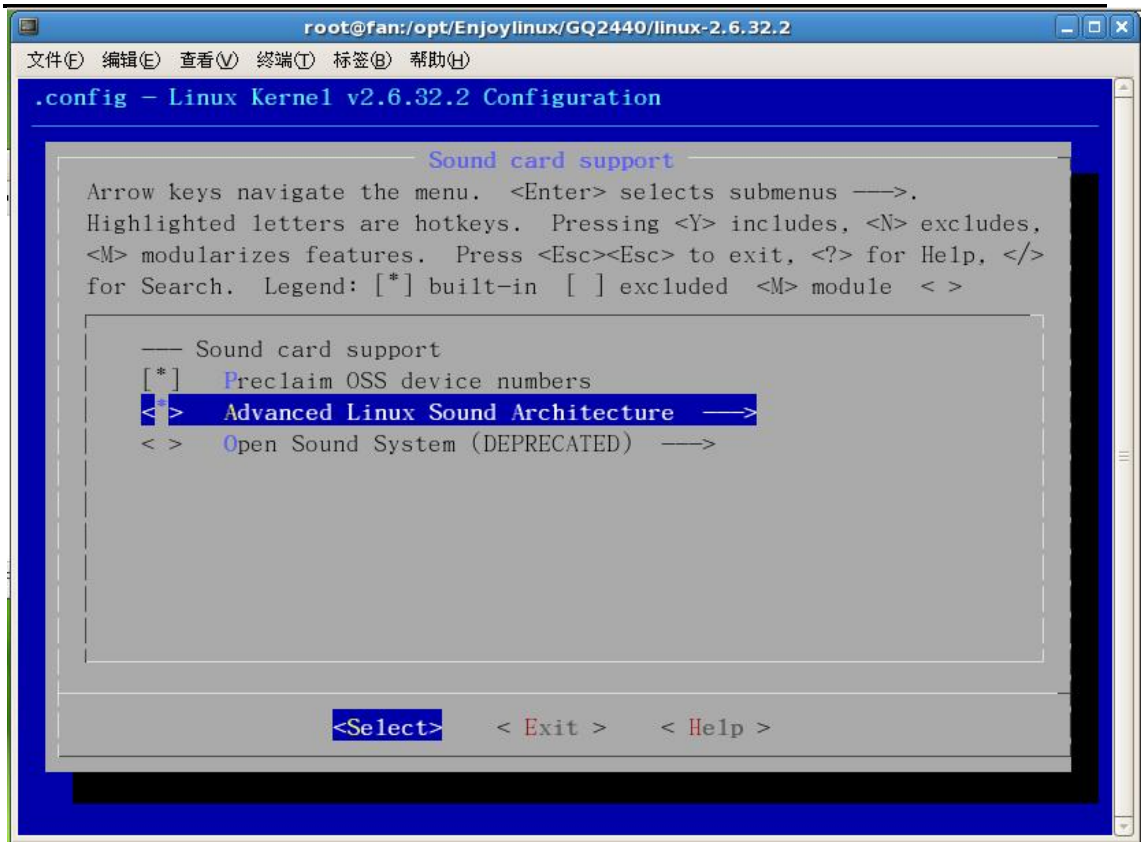
选择<Exit>一直返回到Device Drivers 菜单。

6.3.10 配置音频驱动

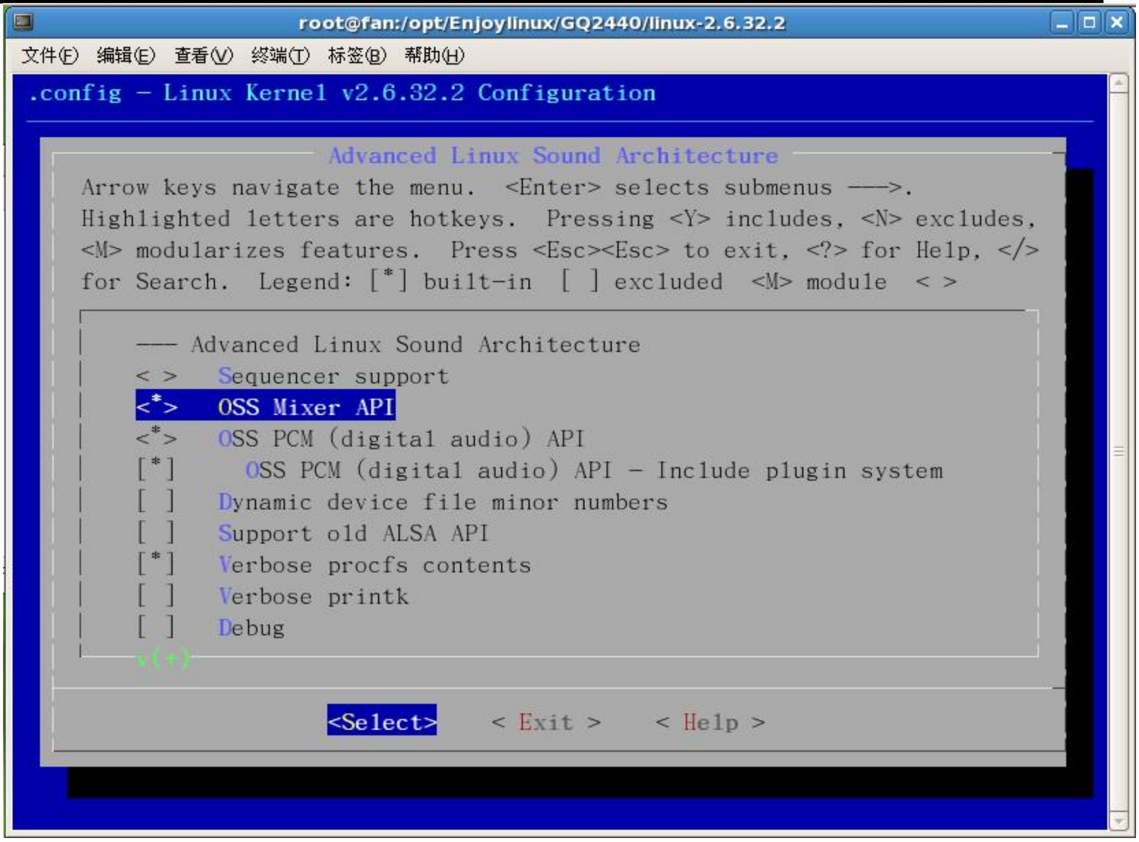
在 Device Drivers 菜单中，选择Sound card supprt，并进入


root@fan:/opt/Enjoylinux/GQ2440/linux-2.6.32.2
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
.config - Linux Kernel v2.6.32.2 Configuration
Device Drivers
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
^(-)
< > Hardware Monitoring support --->
< > Generic Thermal sysfs driver --->
[*] Watchdog Timer Support --->
Sonics Silicon Backplane --->
Multifunction device drivers --->
[ ] Voltage and Current Regulator Support --->
<*> Multimedia support --->
Graphics support --->
<*> Sound card support --->
[*] HID Devices --->
v(+)
<Select> < Exit > < Help >

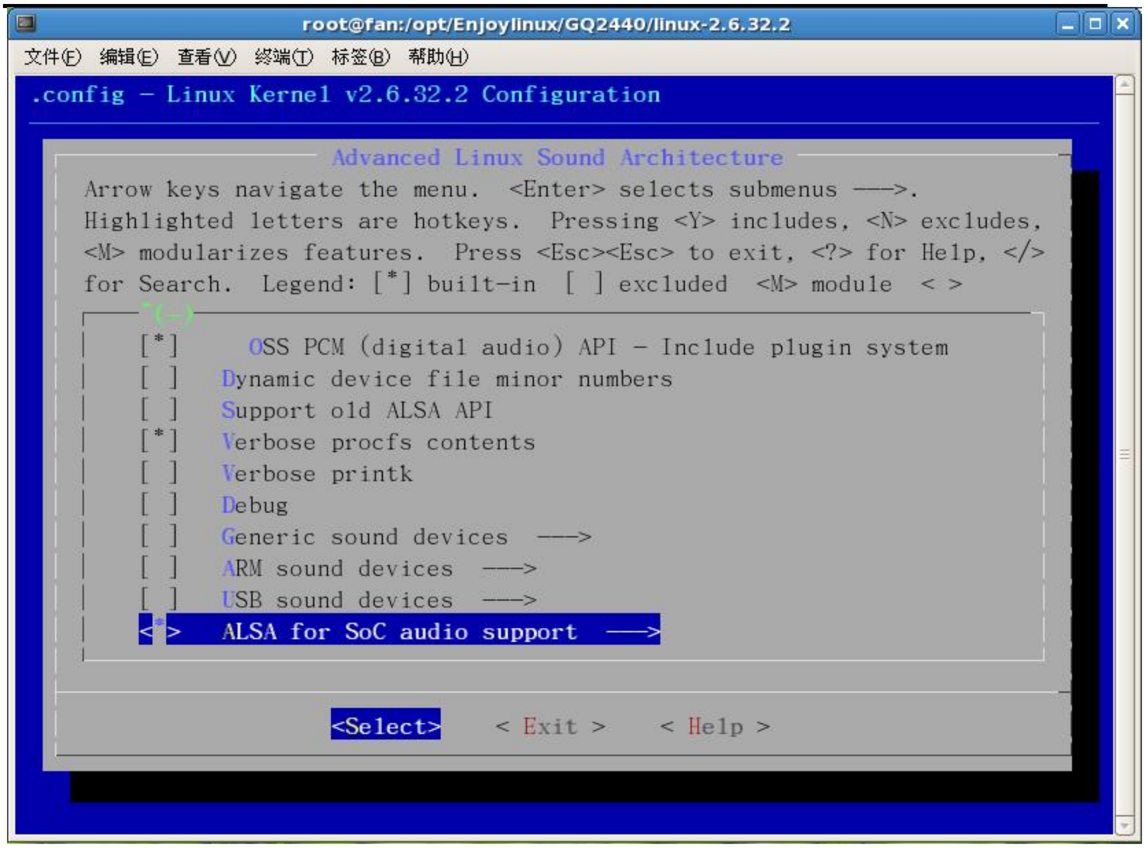
再作如图选择，并进入



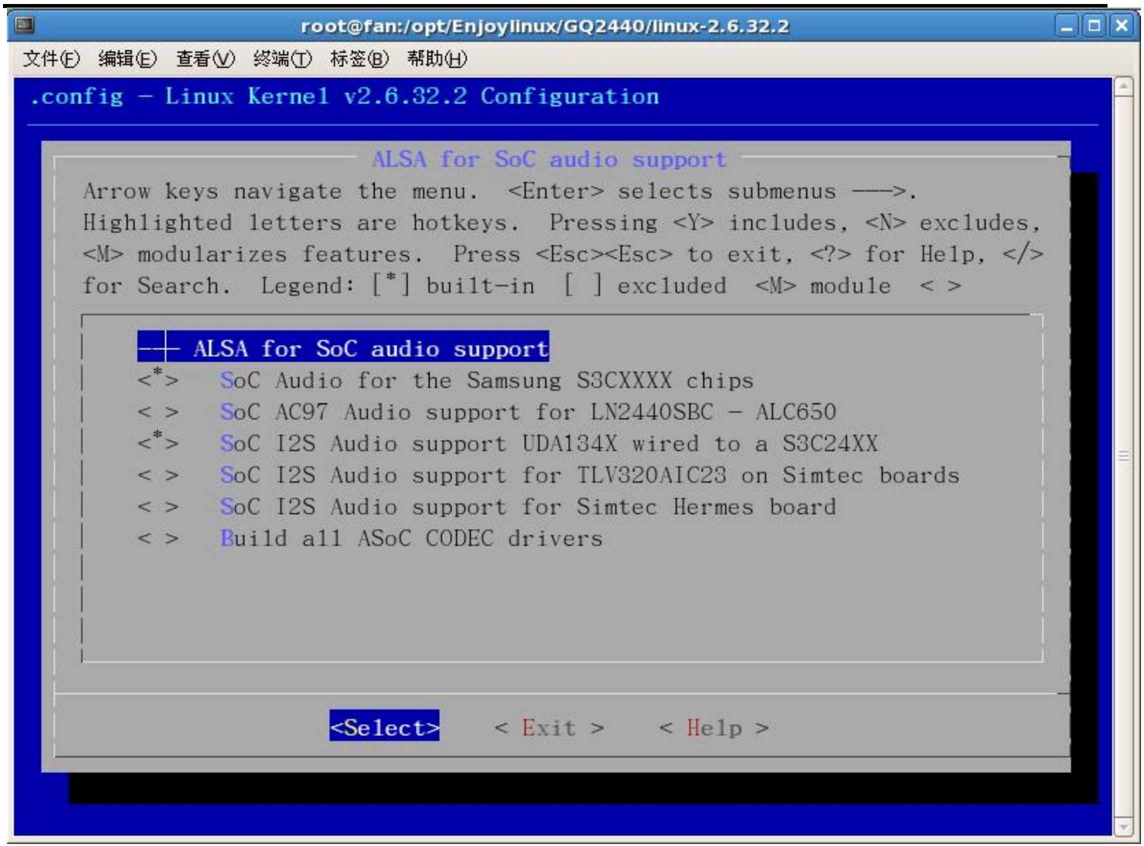
在出现的菜单中，选择 ALSA 接口支持(Advanced Linux Sound Architecture)，并进入



选择 ALSA for Soc audio support，并进入

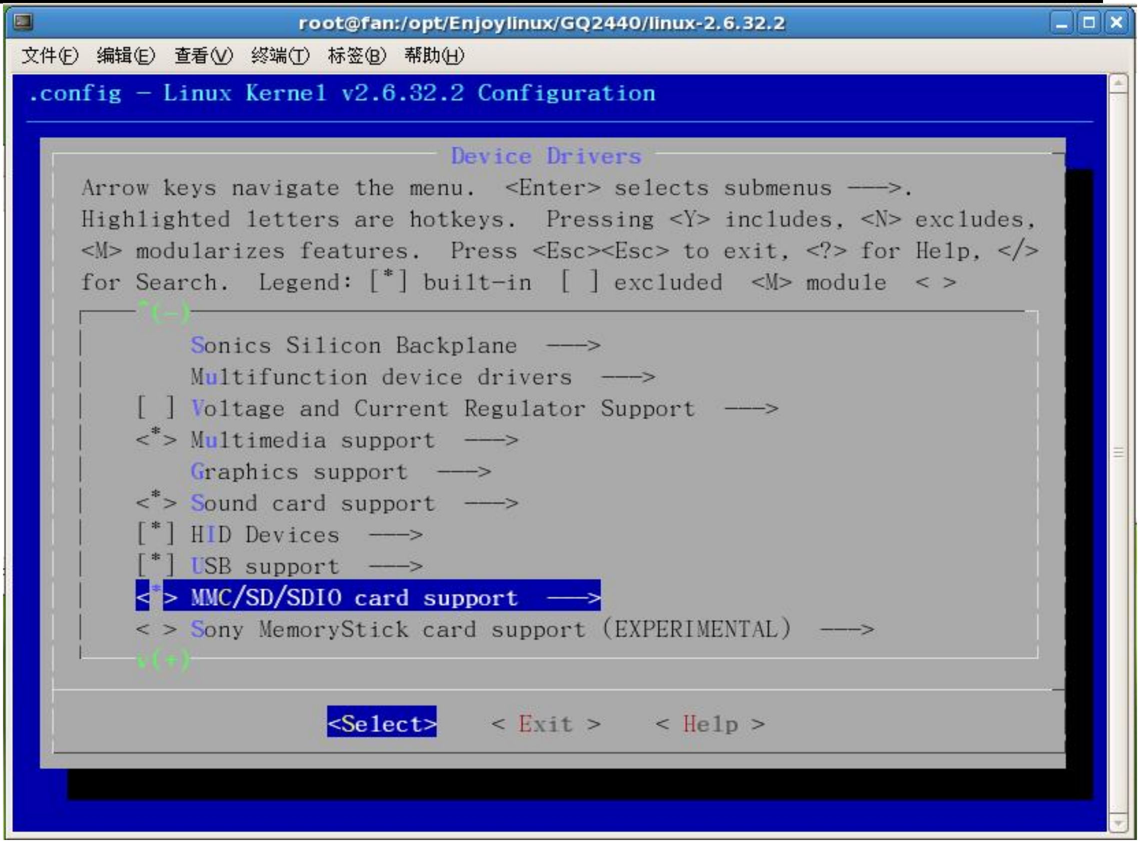


选择 ALSA 接口驱动支持，如图

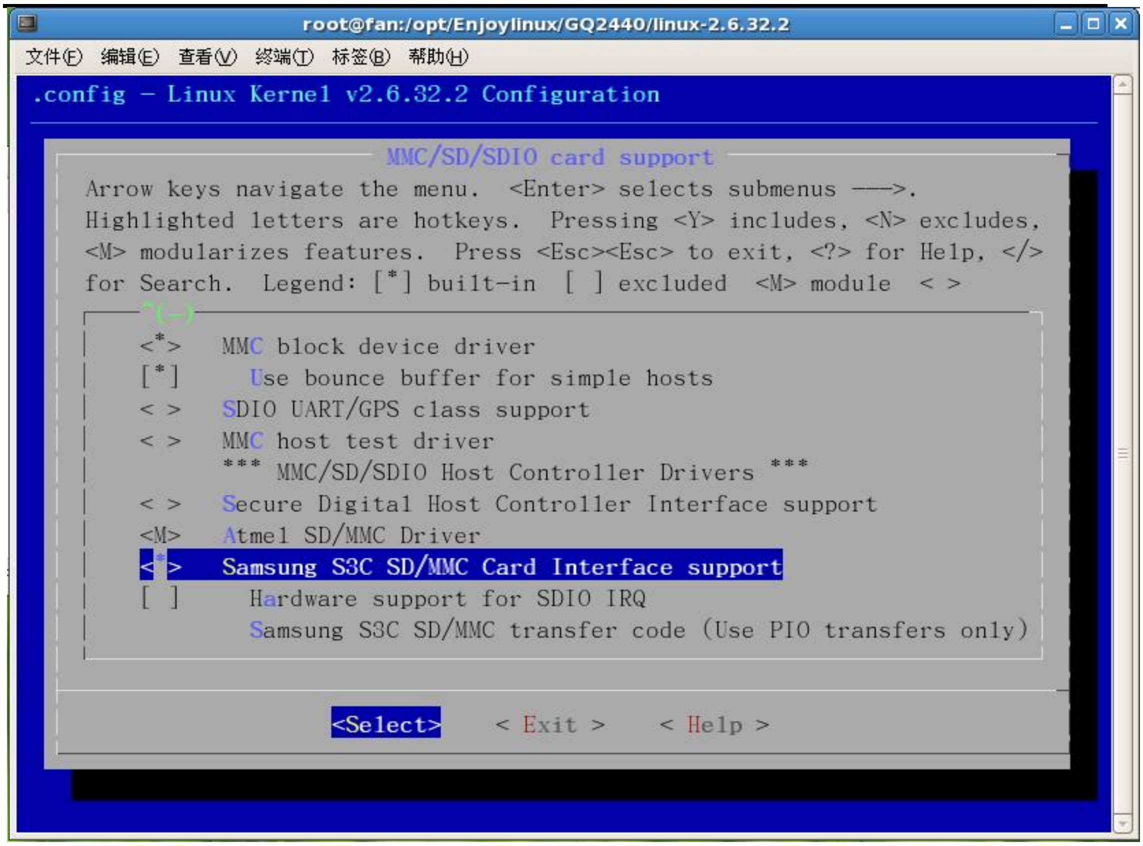


6.3.11 配置 SD/MMC 卡驱动

在 Device Drivers 菜单中，选择 SD/MMC 设备选项并按回车进入



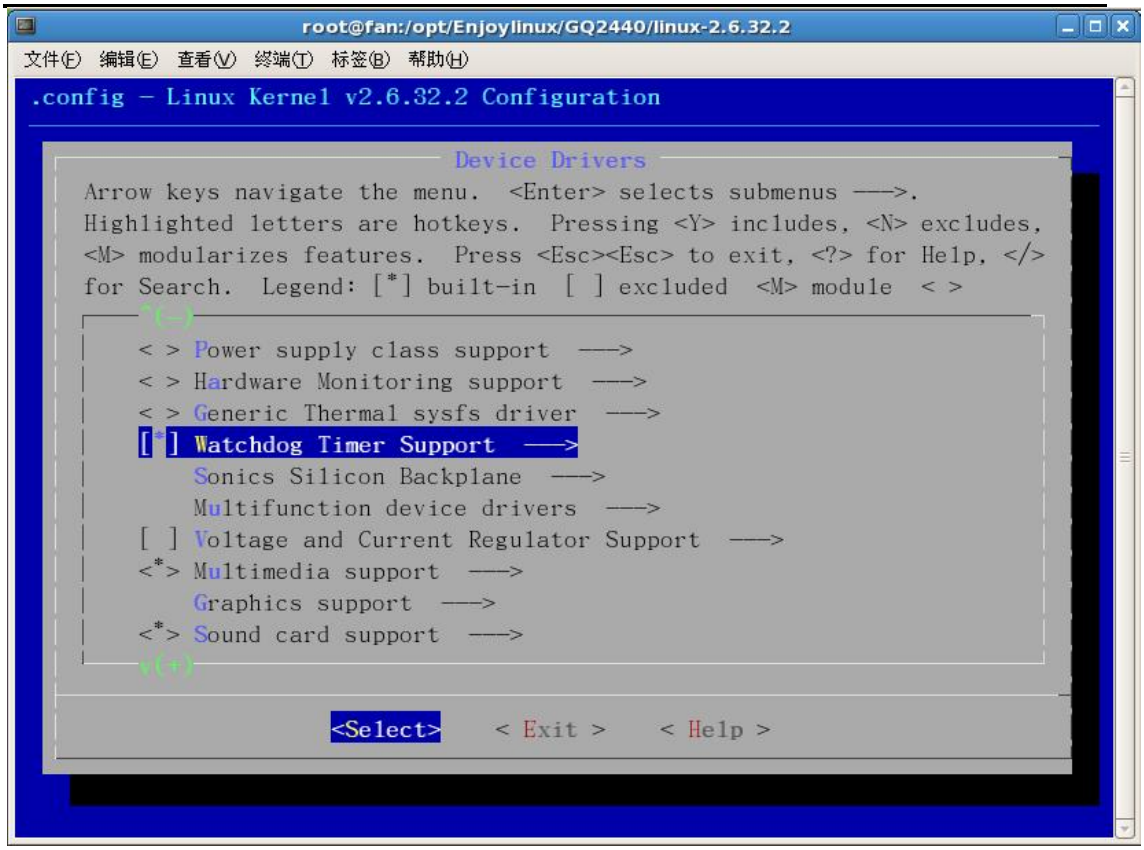
选择如图<*>各项，如图，这样就配置好了 MMC/SD 卡驱动，它可以支持高速大容量 SD 卡，最大可达到 32G:



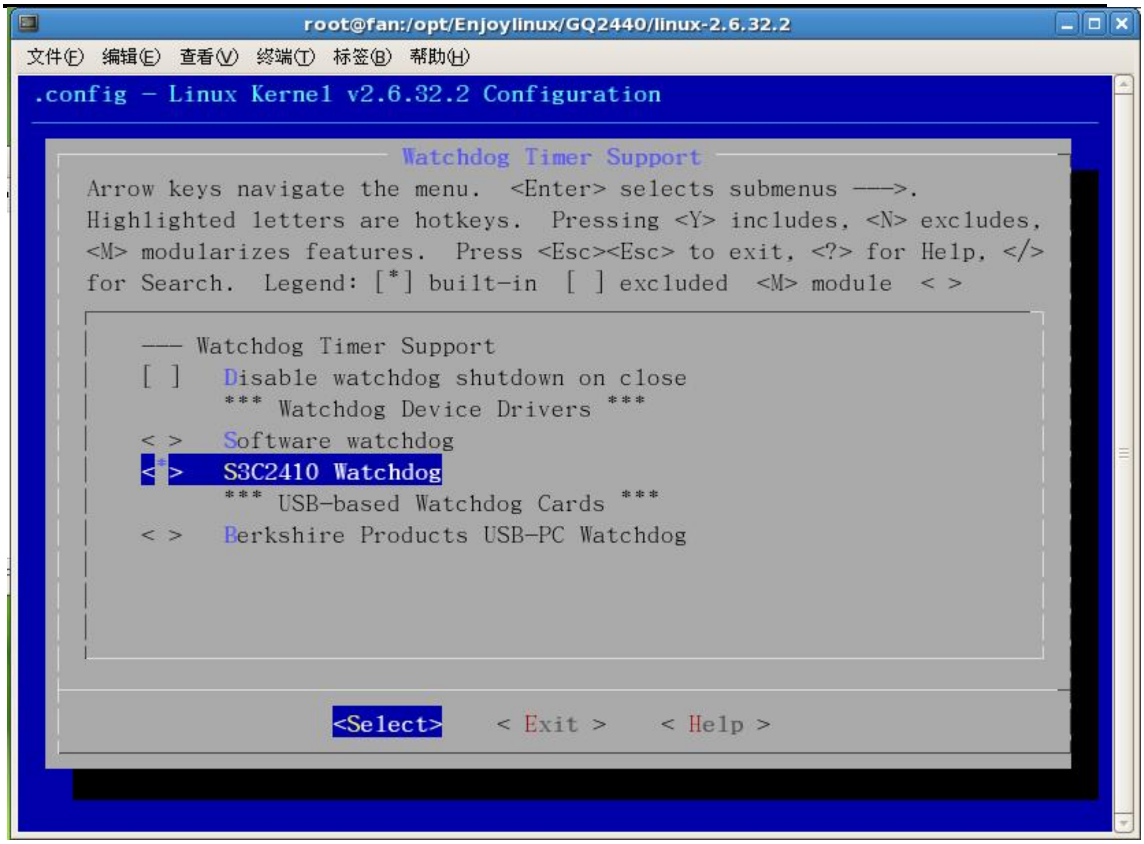
按<Exit>返回到 Device Drivers 菜单。

6.3.12 配置看门狗驱动支持

在 Device Drivers 菜单中，选择 Watchdog 选项并按回车进入



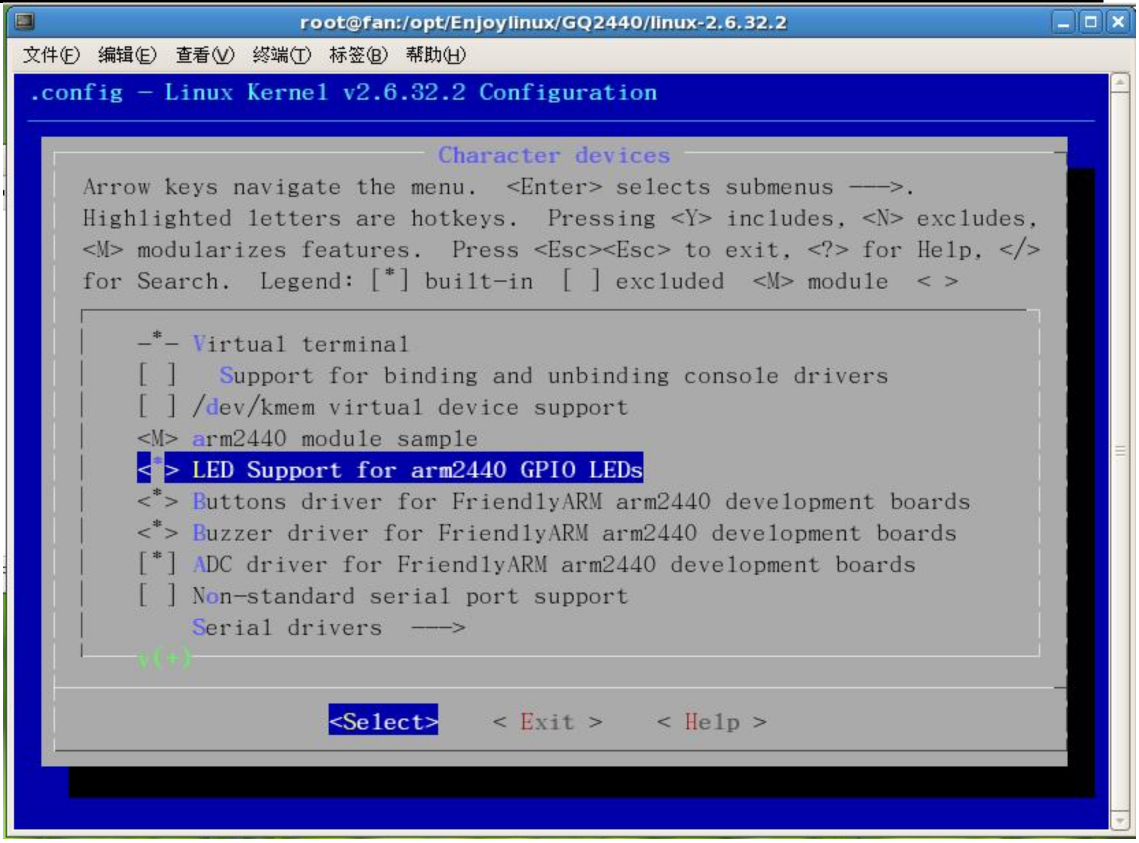
选中如图所示看门狗驱动支持



按<Exit>返回到 Device Drivers 菜单。

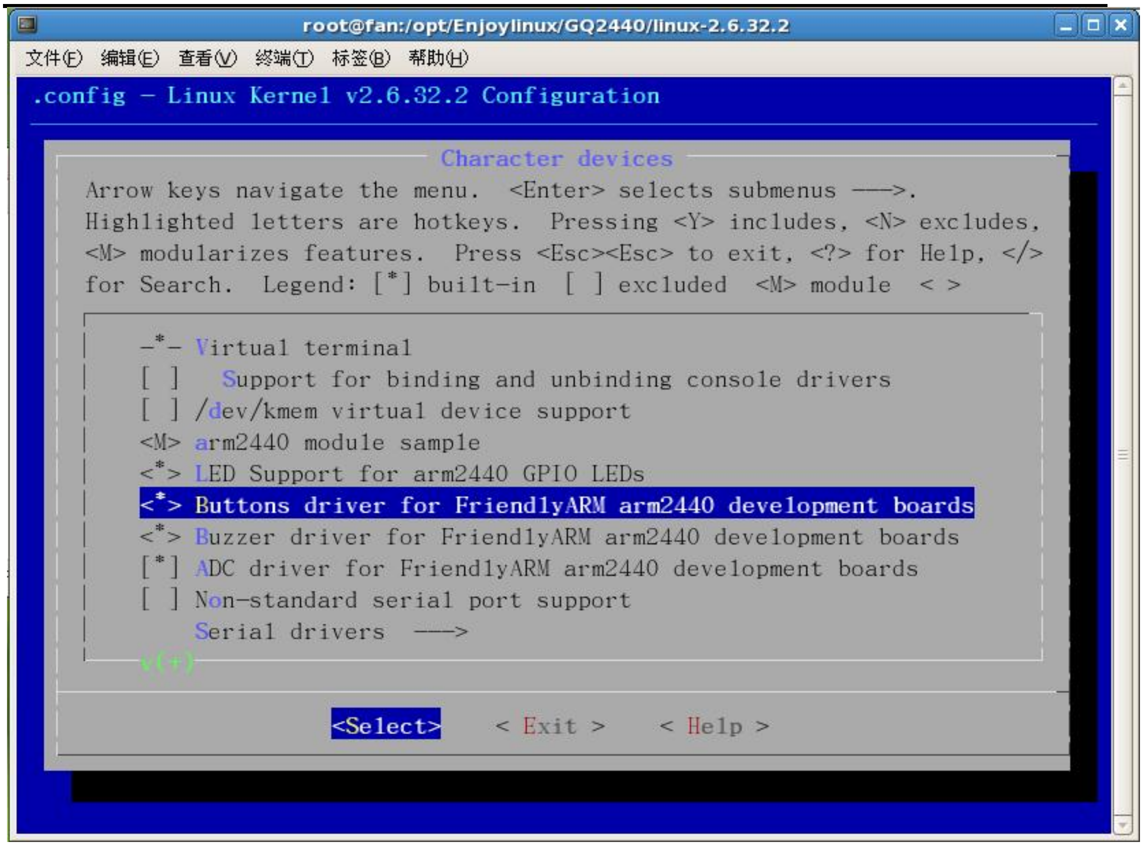
6.3.13 配置 LED 驱动

在 Device Drivers 菜单中，选择进入 Character devices - - ->，找到并选中 LEDs 驱动支持，如图。



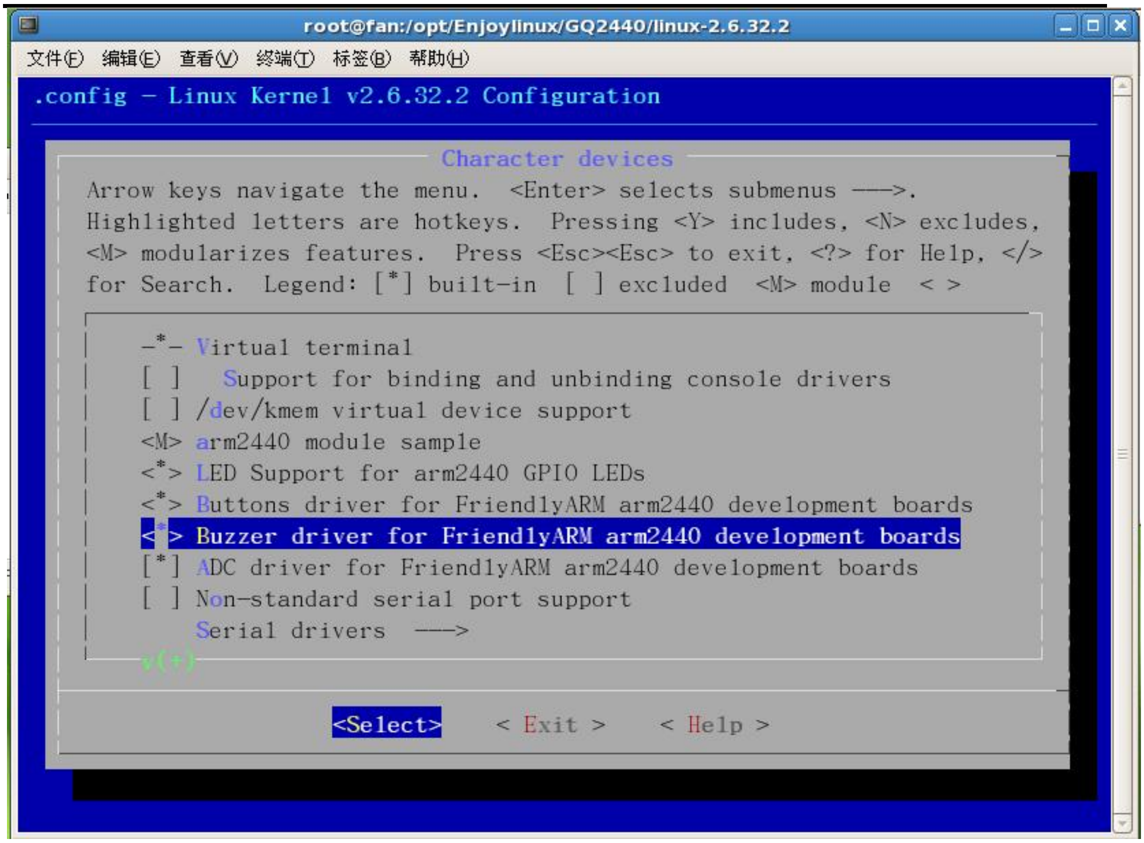
6.3.14 配置按键驱动

在 Device Drivers 菜单中，选择进入 Character devices - - ->，找到并选中 Buttons 驱动支持，如图。



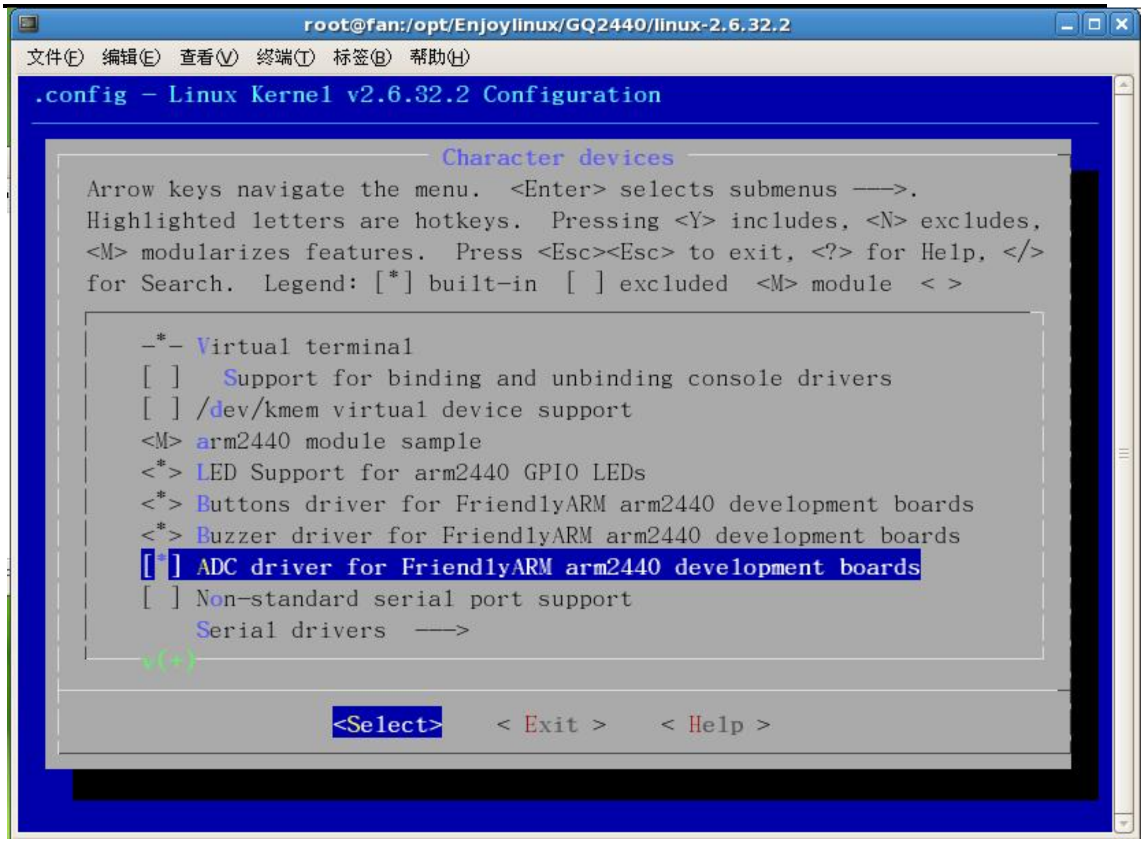
6.3.15 配置 PWM 控制蜂鸣器驱动

依然在 Character devices 菜单中，找到并选中 buzzer 选项，如图



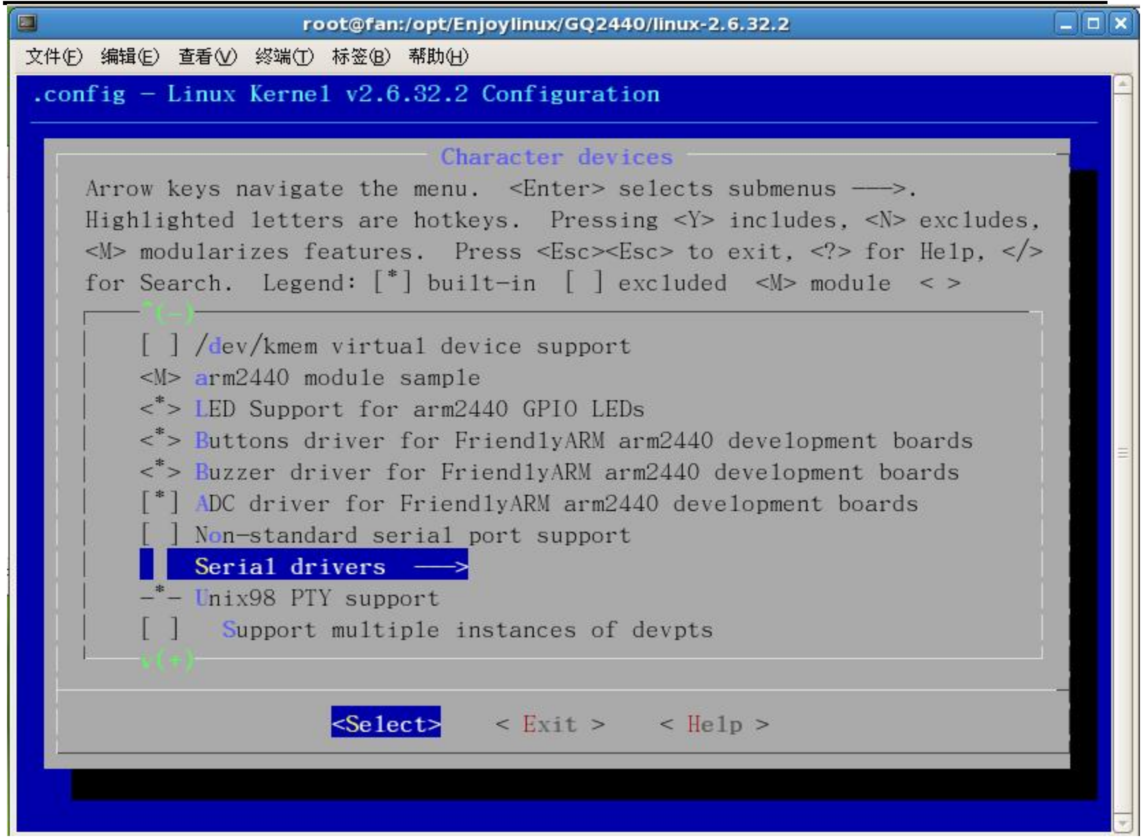
6.3.16 配置 AD 转换驱动

依然在 Character devices 菜单中，找到并选中 ADC 选项，如图

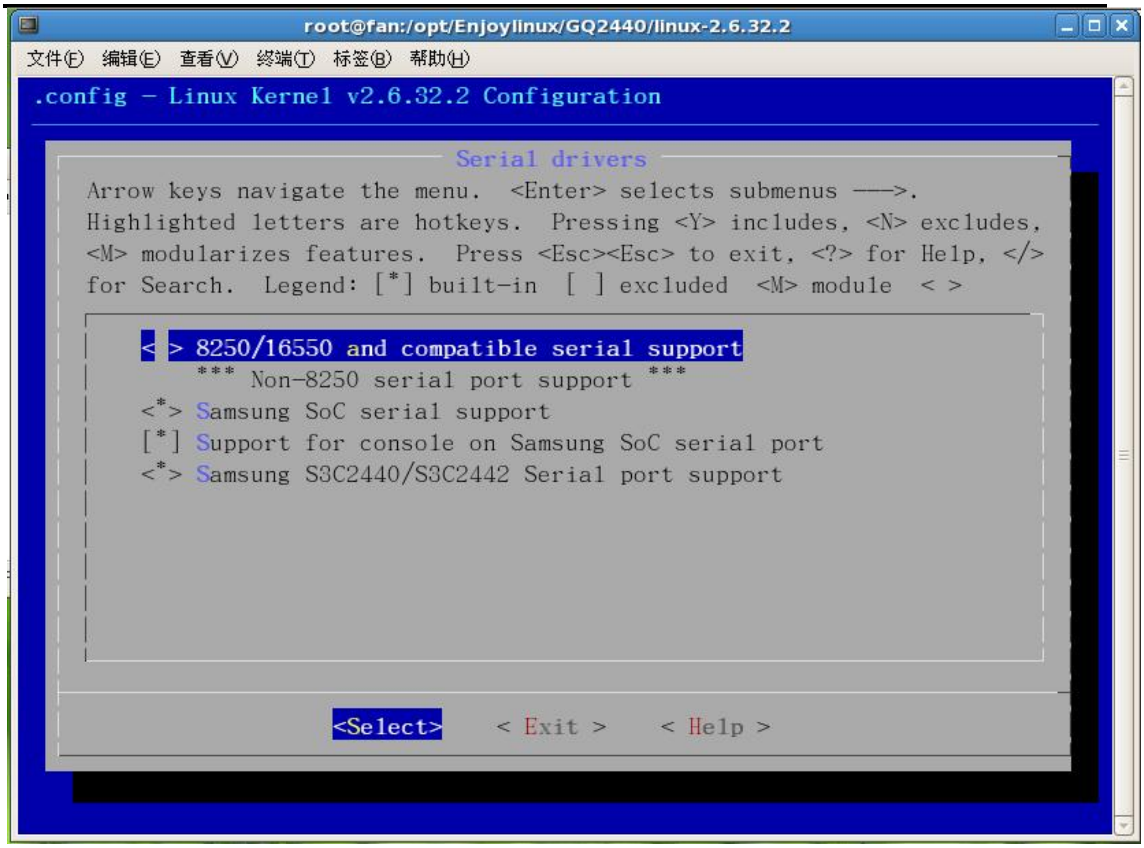


6.3.17 配置串口驱动

依然在 Character devices 菜单中，选择进入 Serial drivers - - ->

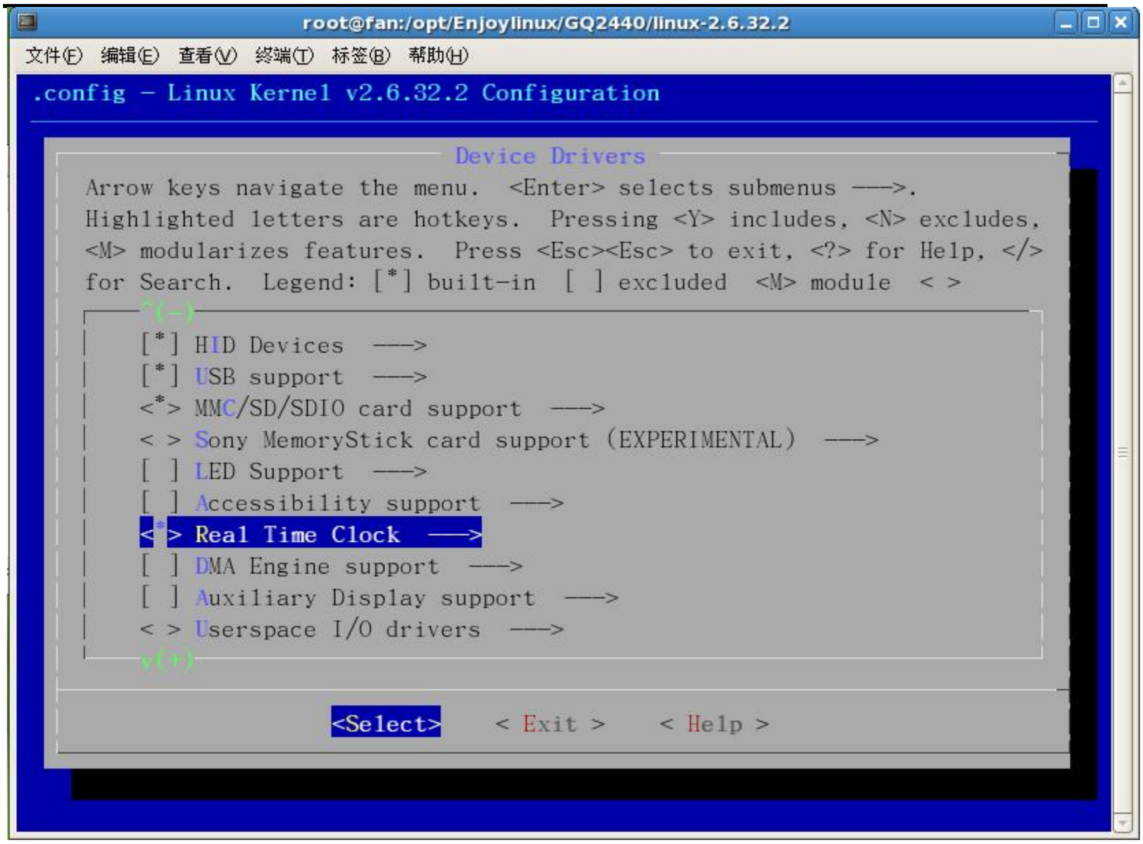


选择如图选项，来配置串口驱动

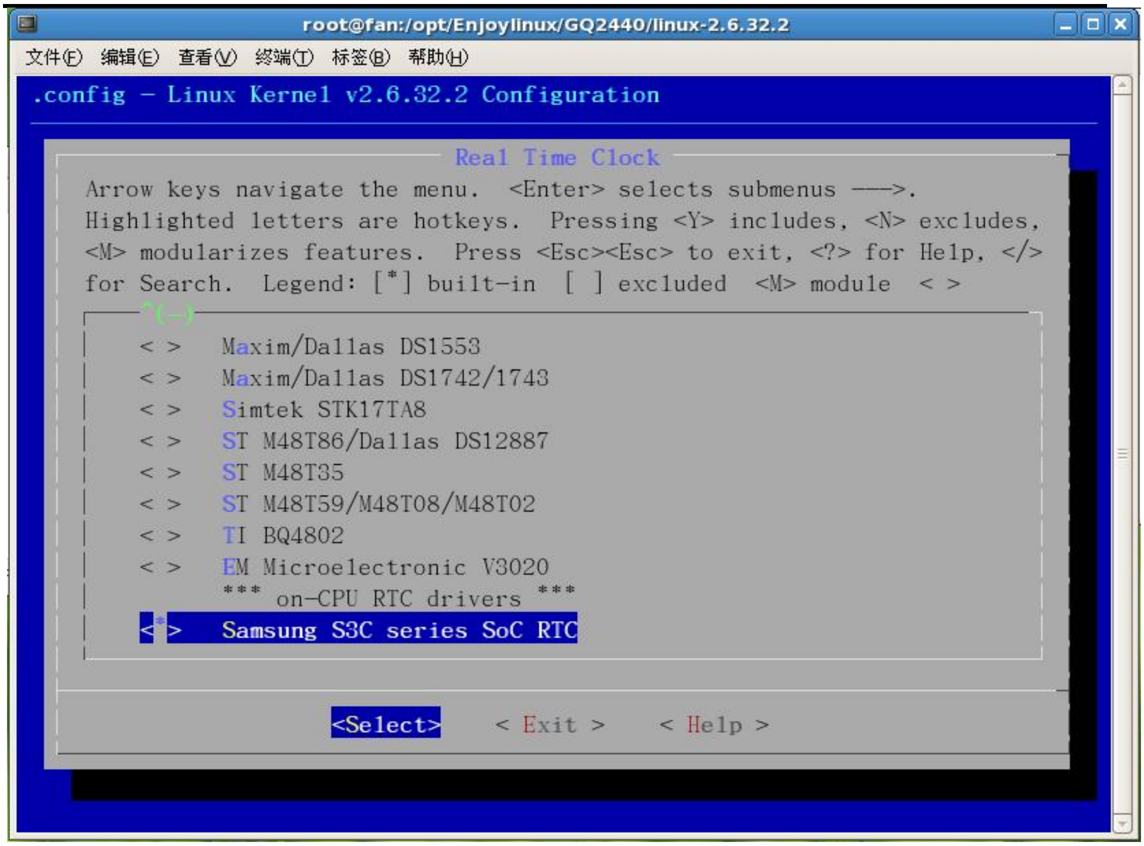


6.3.18 配置 RTC 实时时钟驱动

依然在 Device Drivers 菜单中，选择 Real Time Clock 选项并进入



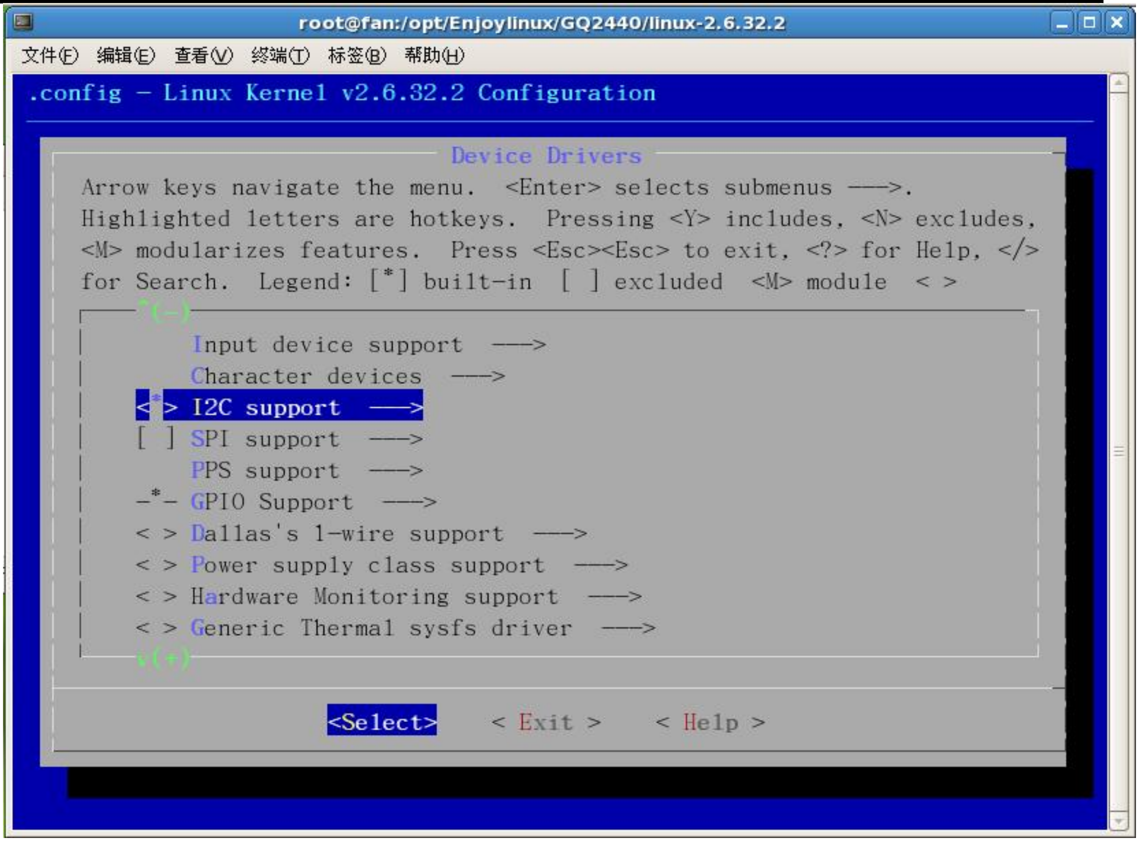
如图选择 2440 系统的 RTC 驱动支持



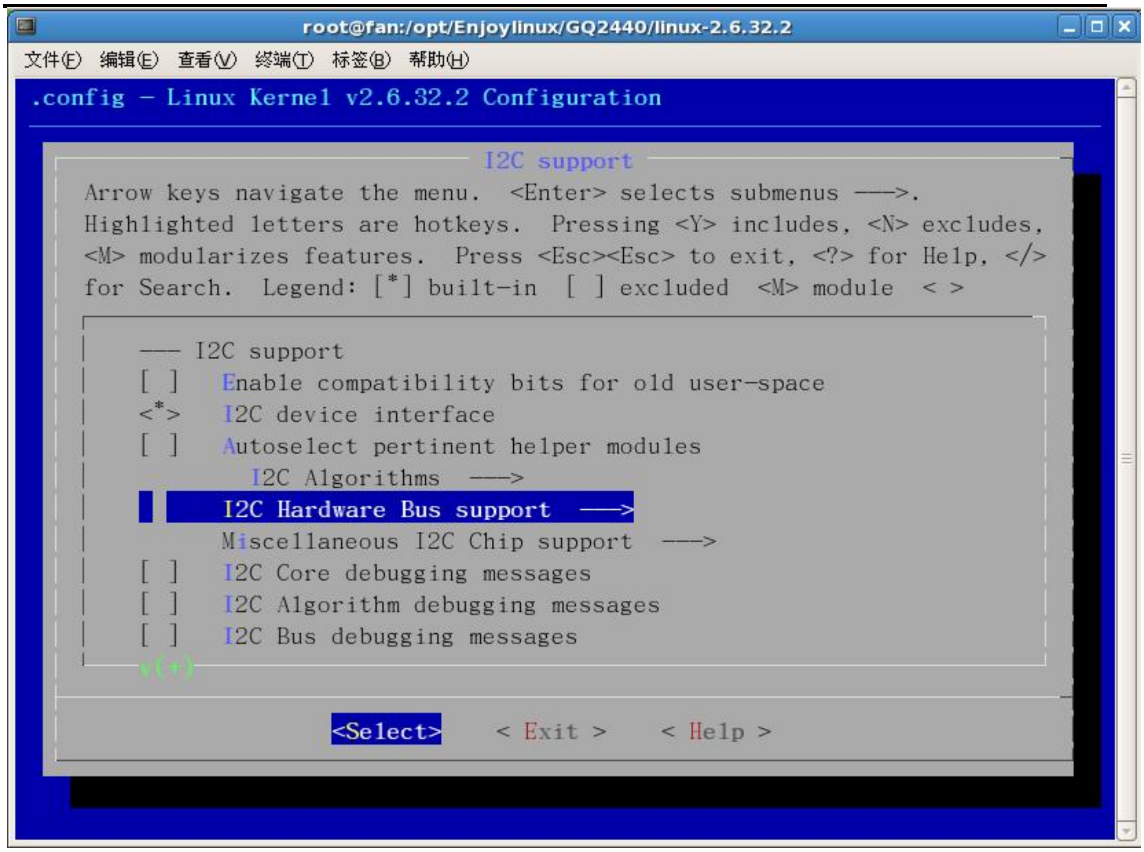
返回到主菜单。

6.3.19 配置 I2C-EEPROM 驱动支持

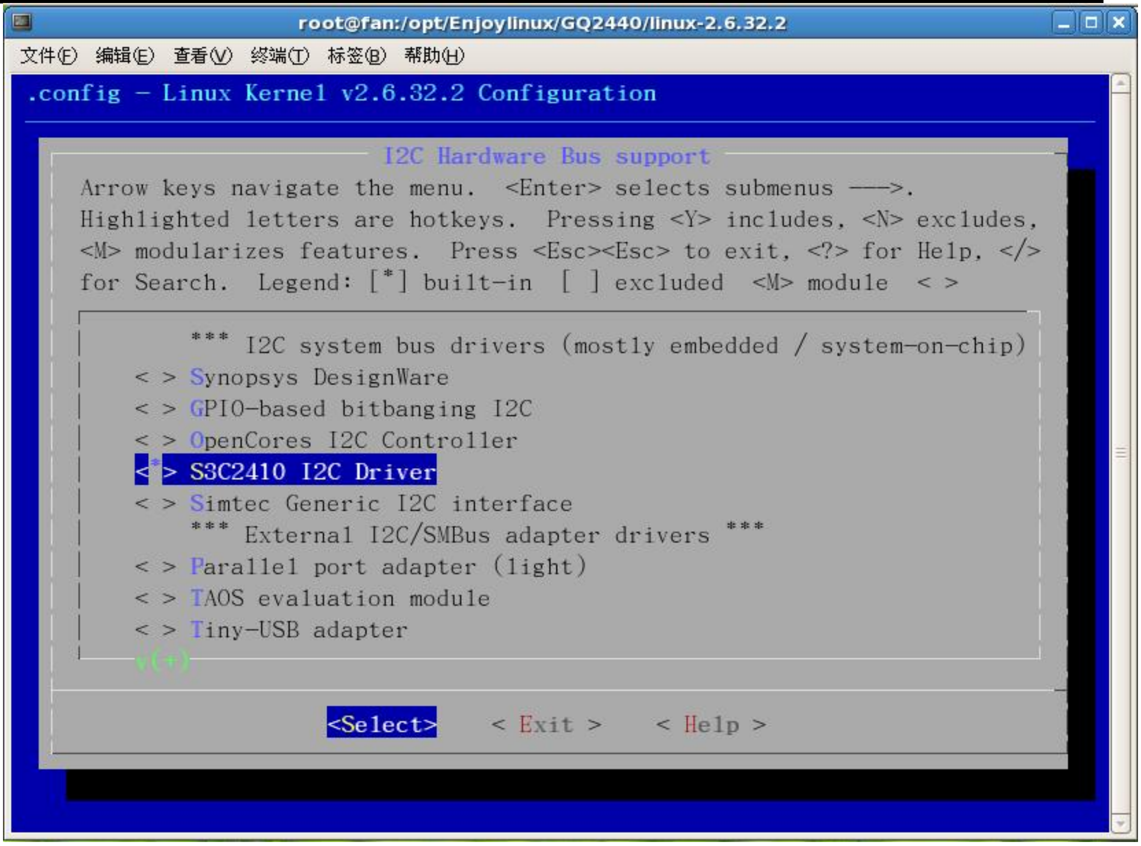
在 Device Drivers 菜单中，找到 I2C support 项，选择进入



在菜单中再选择如图，并进入 I2C Hardware Bus support 子项

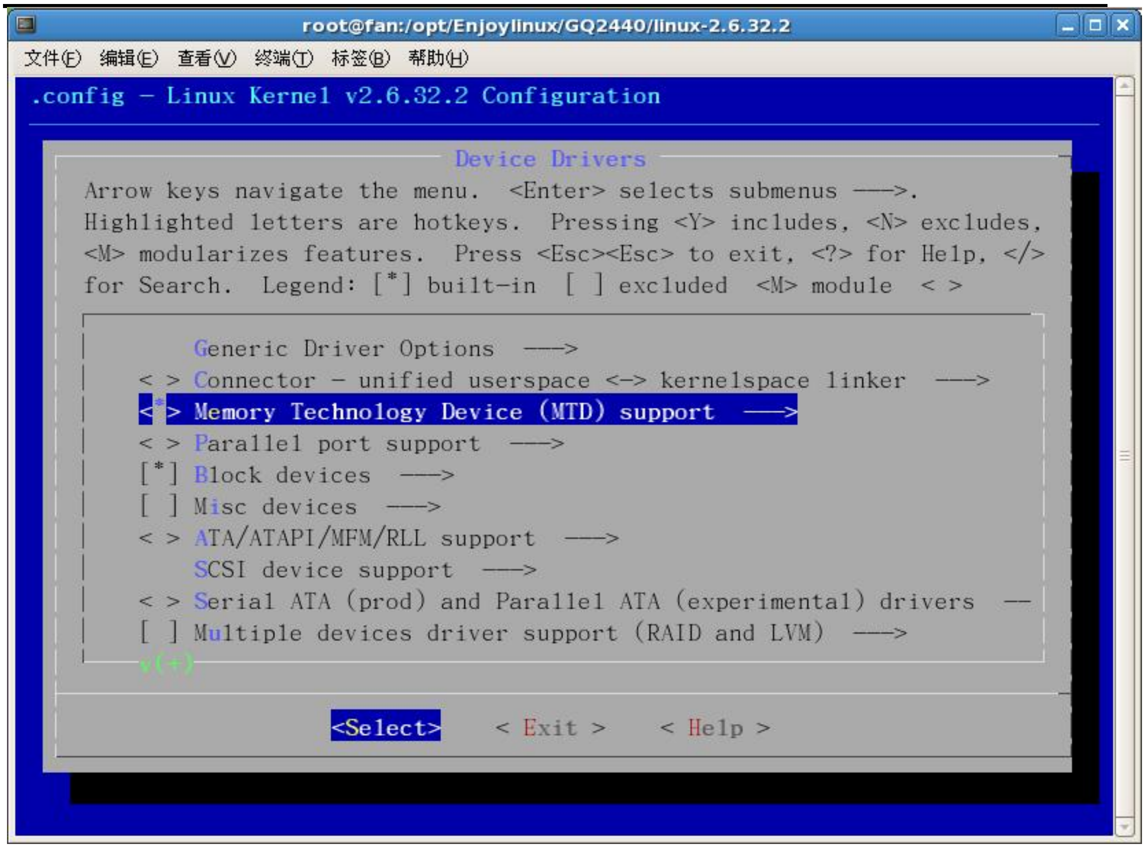


再选择 S3C2410 I2C Driver 即可，如图

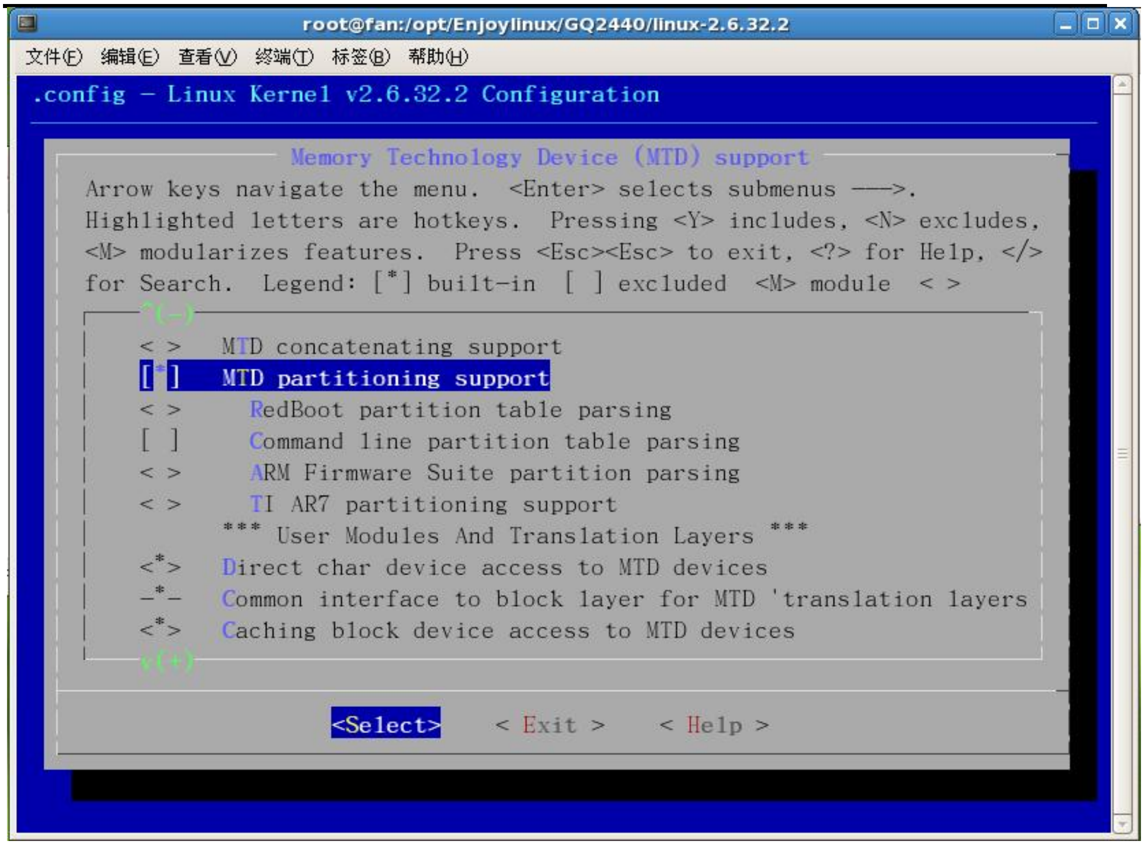


6.3.20 配置 yaff2s 文件系统的支持

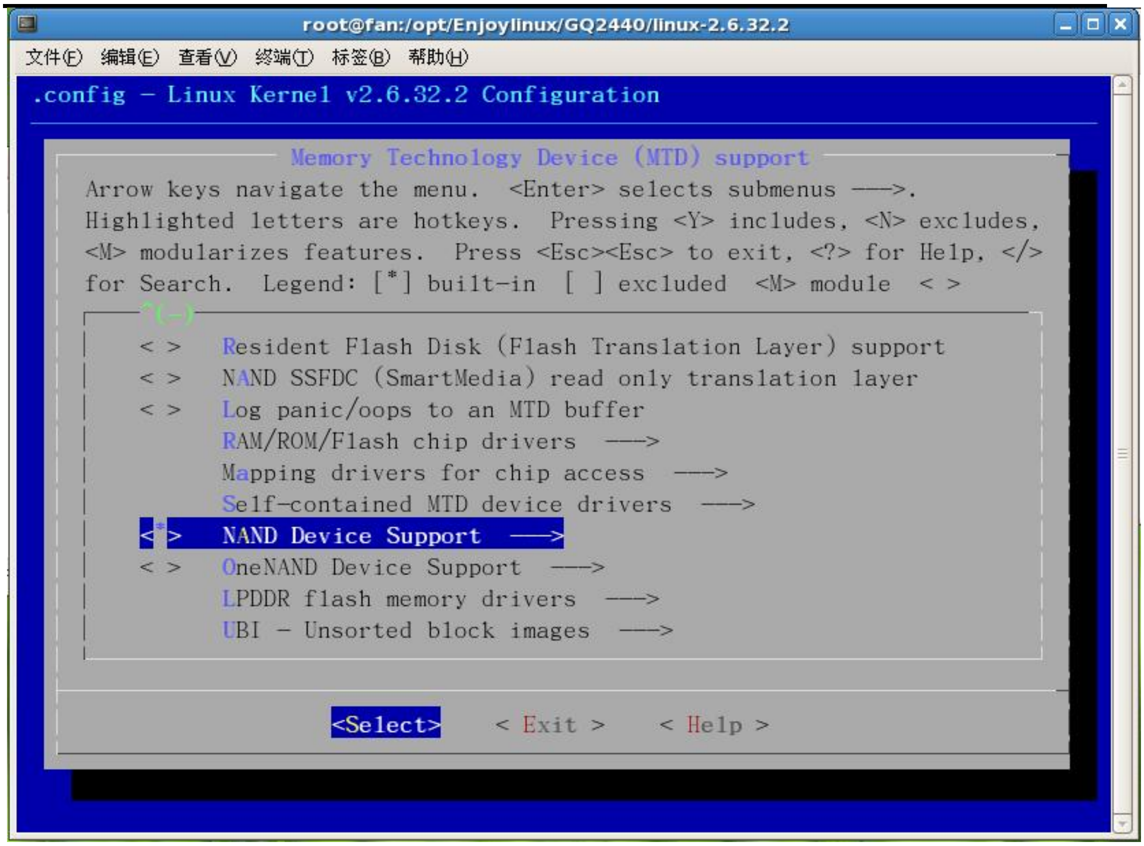
要使用 yaffs2 文件系统，需要先配置 nand flash 驱动支持，在 Device drivers 菜单中选择 MTD 选项如图，并按回车进入



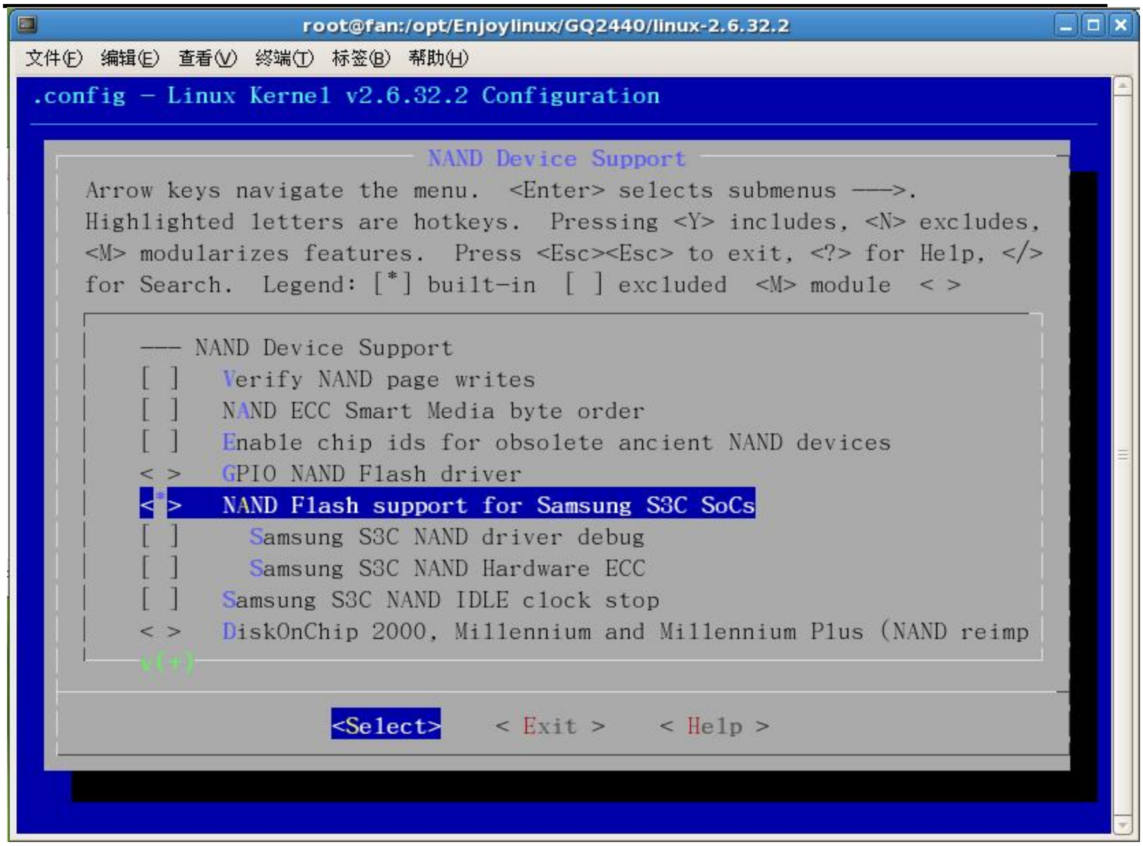
注意子菜单中<*>号的选项，不要取消



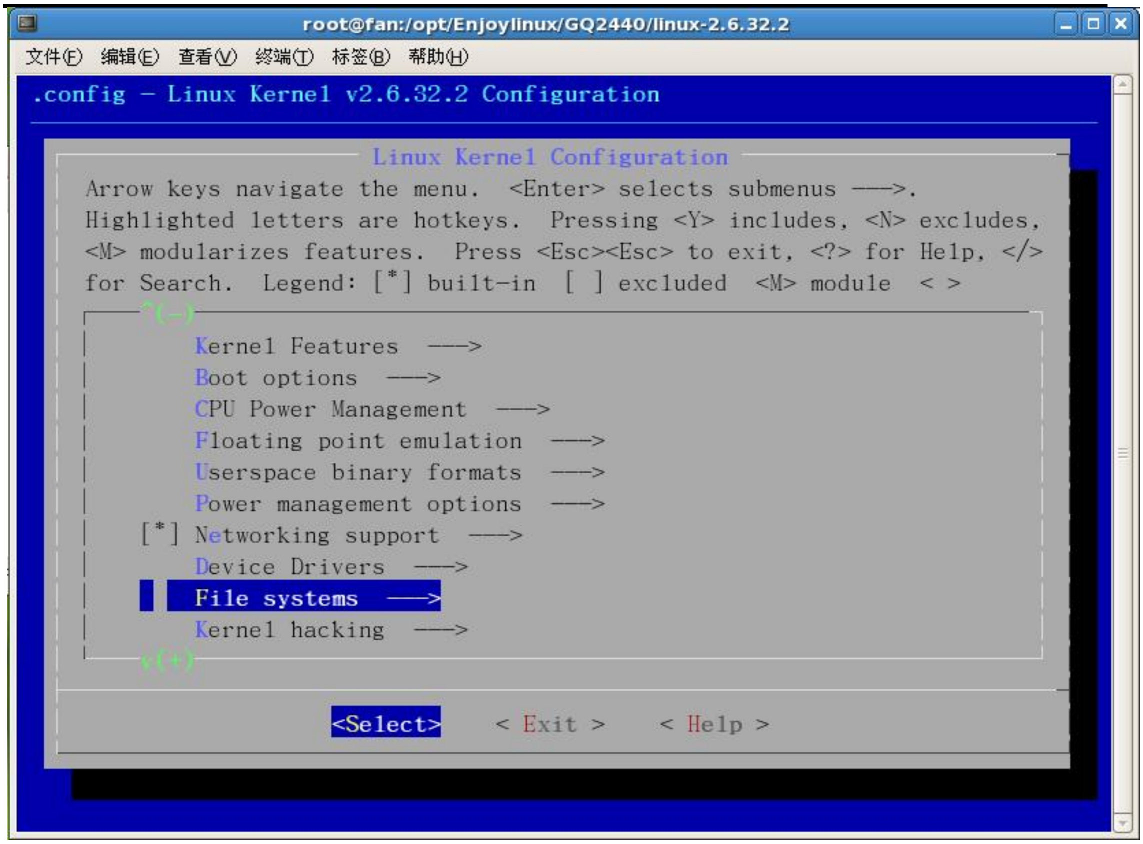
找到 NAND Device Support 选项并进入



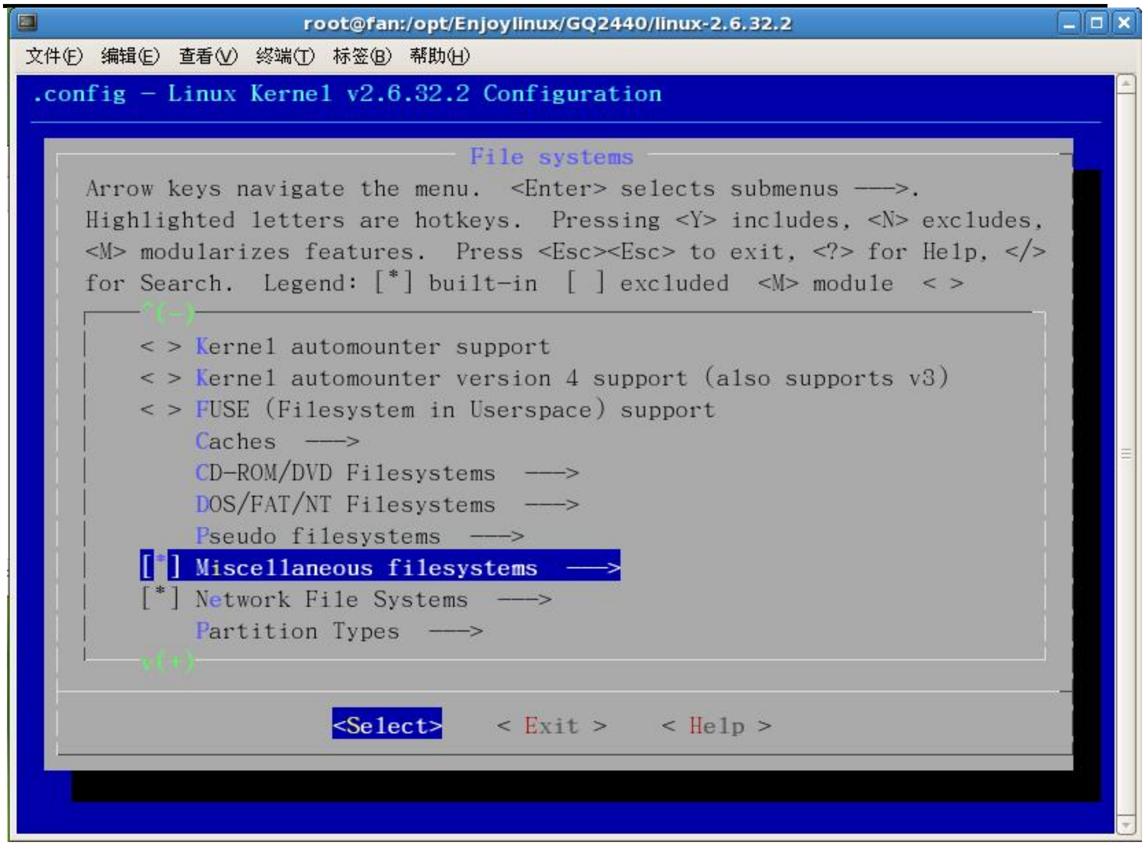
如图选择 Nand Flash 驱动支持



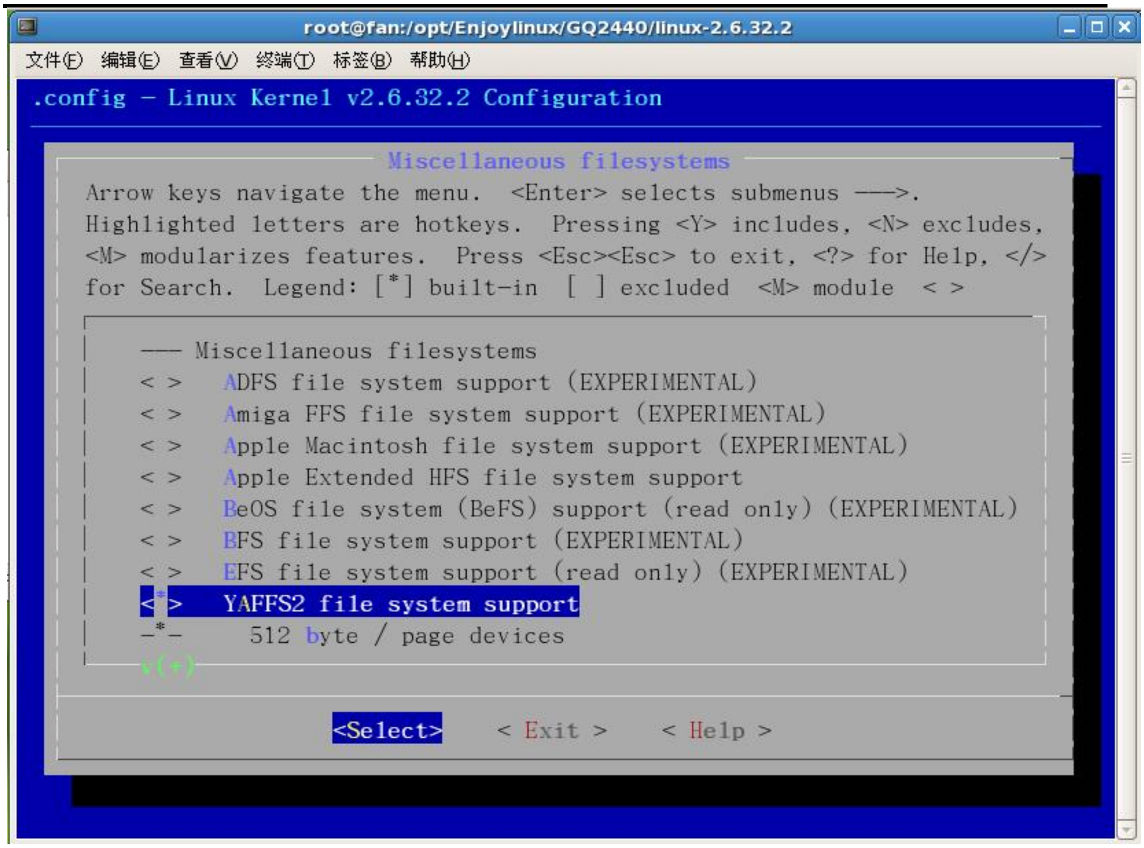
返回到内核配置主菜单，并找到 File systems 选项进入



找到如图选项 Miscellaneous filesystems 并进入



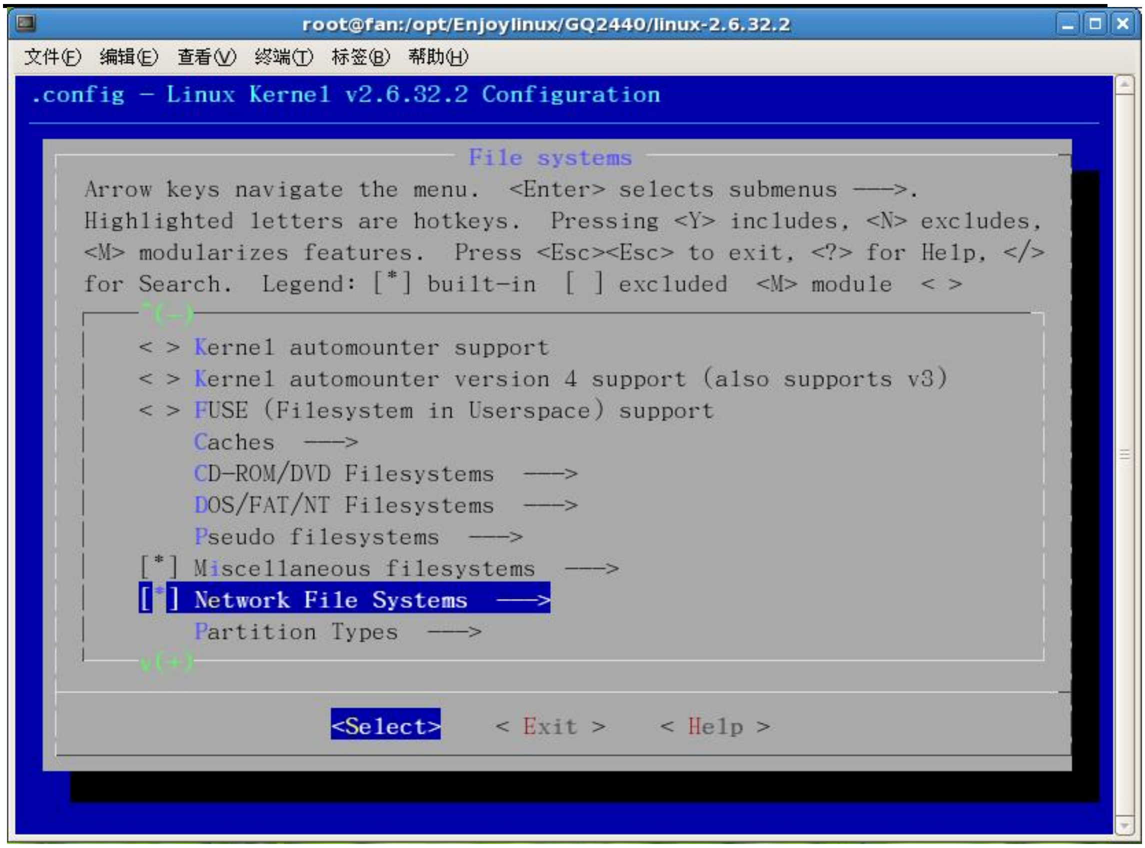
找到 YAFFS2 支持选项，如图选择



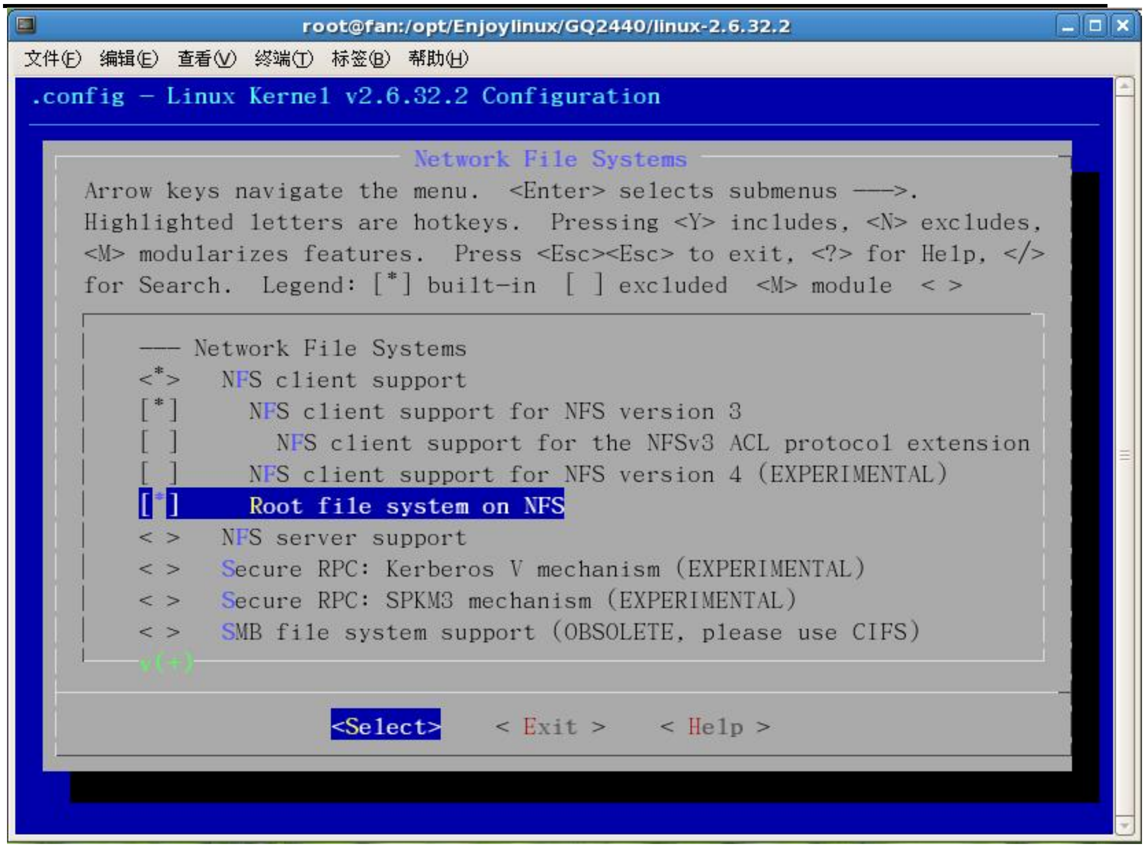
然后<Exit>返回到 File systems 菜单进行下一步

6.3.21 配置 EXT2/VFAT/ NFS 等文件系统

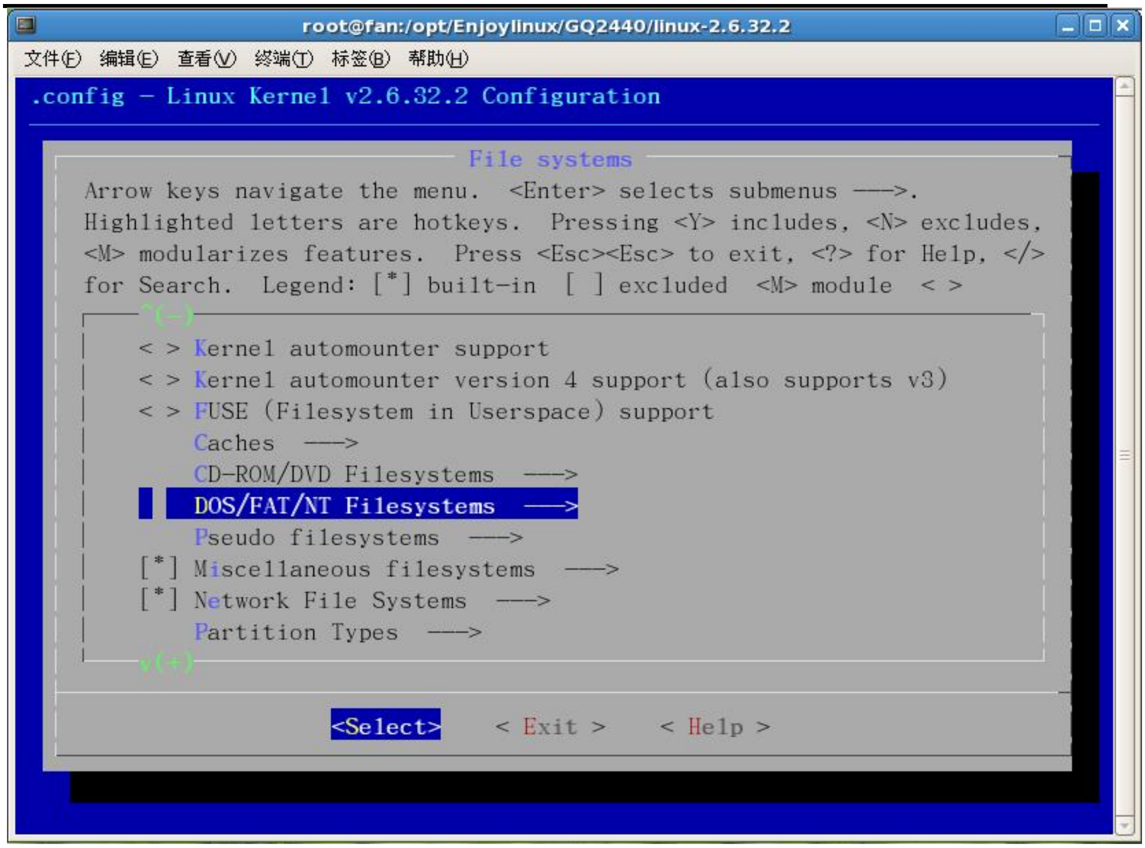
在 File System 菜单中，如图选择 Network File Systems 文件系统的支持并进入。



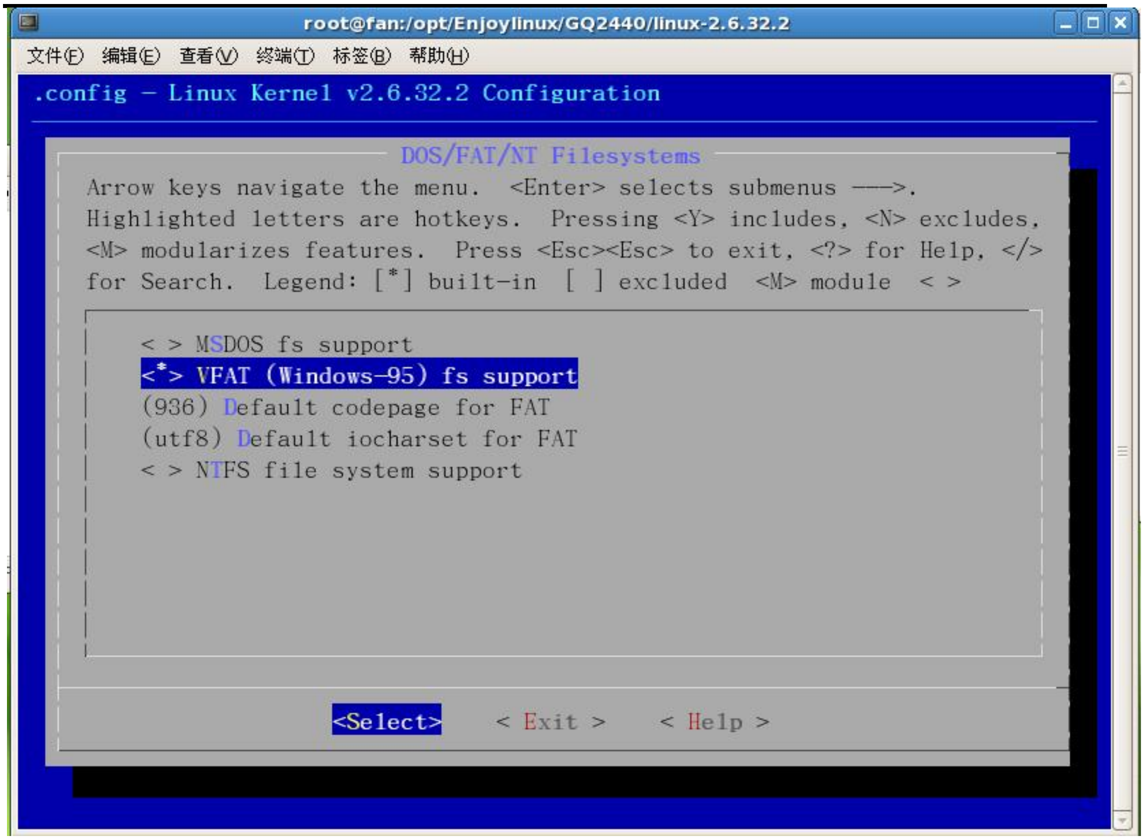
选择如图选项，这样配置编译出的内核就可以通过 NFS 启动系统了。



为了支持优盘或者 SD 卡等存储设备常用的 FAT32 文件系统，还需要配置与此相关的文件系统支持，如图，在 File Systems 菜单中选择 DOS/FAT/NT Filesystems 选项并进入:



在此我们选择了常用的 VFAT 文件系统格式，它可以支持 FAT32:



返回到内核配置主菜单，至此，您已经了解内核的大部分常用选项的配置，更多的内核选项需要您在学习中逐步实践和摸索。

6.4 制作目标板文件系统映象

注意：请务必先按照 5.4.3 章节的介绍安装 mkyaffs2image 文件系统制作工具。

在此，我们以制作测试用的 root_qtopia.img 文件系统映象为例，来介绍 yaffs2 文件系统映象的制作。

进入**/opt/Enjoylinux/GQ2440** 工作目录，执行以下命令：

#cd /opt/Enjoylinux/GQ2440

#mkyaffs2image root_qtopia root_qtopia.img

提示：如果你使用的是 128M/256M/512M/1GB Nand Flash 的 GQ2440，请使用 mkyaffs2image-128M 制作工具，在此不再赘述。

如图：

```
root@fan:/opt/Enjoylinux/GQ2440
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
Object 3085, root_qtopia//usr/local/lib/ts/linear.so is a file, 3 data chunks wr
itten
Object 3086, root_qtopia//usr/local/lib/ts/input.so is a file, 3 data chunks wri
tten
Object 3087, root_qtopia//usr/local/lib/libts-0.0.so.0 is a symlink to "libts-0.
0.so.0.1.1"
Object 3088, root_qtopia//usr/local/bin is a directory
Object 3089, root_qtopia//usr/local/bin/ts_calibrate is a file, 12 data chunks w
ritten
Object 3090, root_qtopia//usr/local/qtopia-titles-cn.tar.gz is a file, 3 data ch
unks written
Object 3091, root_qtopia//usr/local/etc is a directory
Object 3092, root_qtopia//usr/local/etc/ts.conf is a file, 1 data chunks written
Object 3093, root_qtopia//usr/local/qtopia-titles-en.tar.gz is a file, 2 data ch
unks written
Object 3094, root_qtopia//usr/local/qtopia-titles-jp.tar.gz is a file, 3 data ch
unks written
Object 3095, root_qtopia//dev is a directory
Object 3096, root_qtopia//dev/console is a character device
Object 3097, root_qtopia//var is a directory
Operation complete.
2841 objects in 255 directories
34748 NAND pages
[root@fan GQ2440]#
```

可以看到，已经在当前目录下生成了 root_qtopia.img 映象文件，使用前面章节介绍的烧写方法，可以通过 USB 把 root_qtopia.img 烧写到目标板。

第7章 嵌入式 Linux 应用开发入门指南

本节内容通过嵌入式 Linux 开发最简单的例子，介绍了如何编写和编译 Linux 应用程序，并下载到开发板运行起来，并介绍了如何制作和装载驱动程序模块，以及移植一些常见的开源软件。

嵌入式 Linux 资源丰富，我们不可能介绍到每一个细节，本文旨在提供一些嵌入式 Linux 经常用到的方法，为你打开奇妙世界的大门。

注意：以下示例程序所使用的编译器为 arm-linux-gcc-4.3.2 with EABI，如果你使用了其他版本的交叉编译器，编译完有可能无法在开发板上运行。要检查交叉编译器的版本类型，可在终端运行命令：arm-linux-gcc –v，如图

```
[root@fan /]# arm-linux-gcc-4.3.2 -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure --build
=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --en
able-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --wit
h-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers
=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72' --wit
h-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/
opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --
with-build-sysroot=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnue
abi/libc --with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-ar
m-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-resp
in/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --
disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scra
tch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-tim
e-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
[root@fan /]#
```

7.1 Hello,World!

7.1.1 Hello,World 源代码

Hello,World 源代码位于光盘的 **Linux 源代码和工具\examples.tgz** 包中，如果您按第五章节的步骤安装了开发环境，它将位于**/opt/Enjoylinux/GQ2440/examples/hello** 目录，其源代码如下：

```
#include <stdio.h>

int main(void) {
	printf("hello, Enjoylinux !\n");
}
```

7.1.2 编译 Hello,World

首先进入测试程序源代码目录

#cd /opt/Enjoylinux/GQ2440/examples/hello

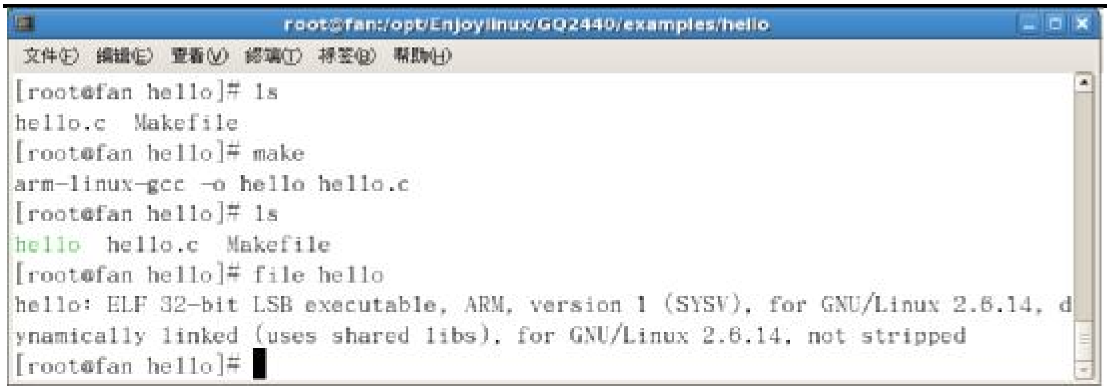
然后，使用命令行进行手工交叉编译示例程序

#arm-linux-gcc –o hello main.c

或者借助编译脚本进行编译

#make

最后将生成 hello 可执行文件，使用 file 命令可以检查你生成的 hello 可执行文件是否为 ARM 体系和格式版本，能在开发板上正常运行的可执行文件一般如图所示：



7.1.3 把 Hello,World 下载到开发板运行

将编译好的可执行文件下载到目标板目前主要四种方式:

第一种：复制到介质(如 U 盘)

第二种：通过网络传送文件到开发板(推荐使用)

第三种：通过串口传送文件到开发板

第四种：通过 NFS(网络文件系统)直接运行

下面分别进行介绍:

(1)使用优盘

方法： 先把编译好的可执行程序复制到 U 盘，再把优盘插到目标板上并挂载它，然后把程序拷贝到目标板执行。

步骤:

Step1: 复制程序到 U 盘

把优盘插到 PC 的 USB 接口，执行以下命令把程序复制到 U 盘

#mount /dev/sda1 /mnt *;挂载 U 盘*

#cp hello /mnt *;复制刚才编译好的程序到 U 盘*

#umount /mnt *;卸载 U 盘*

Step2: 把程序从 U 盘拷贝到目标板并执行

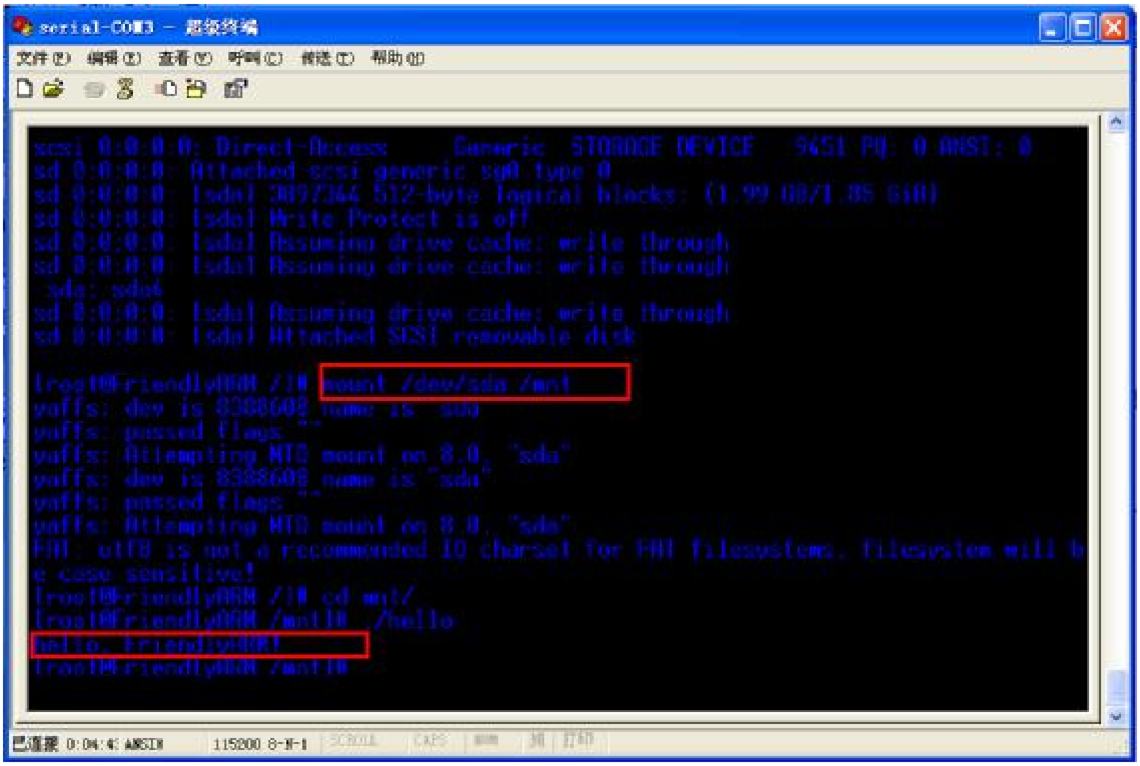
把优盘插入到开发板的 USB Host 接口，将 U 盘挂载到开发板的/mnt 目录下，执行以下命令就可以运行 hello 程序了。

#mount /dev/sda4 /mnt *;挂载 U 盘*

#cd /mnt

#./hello *;执行 hello 程序*

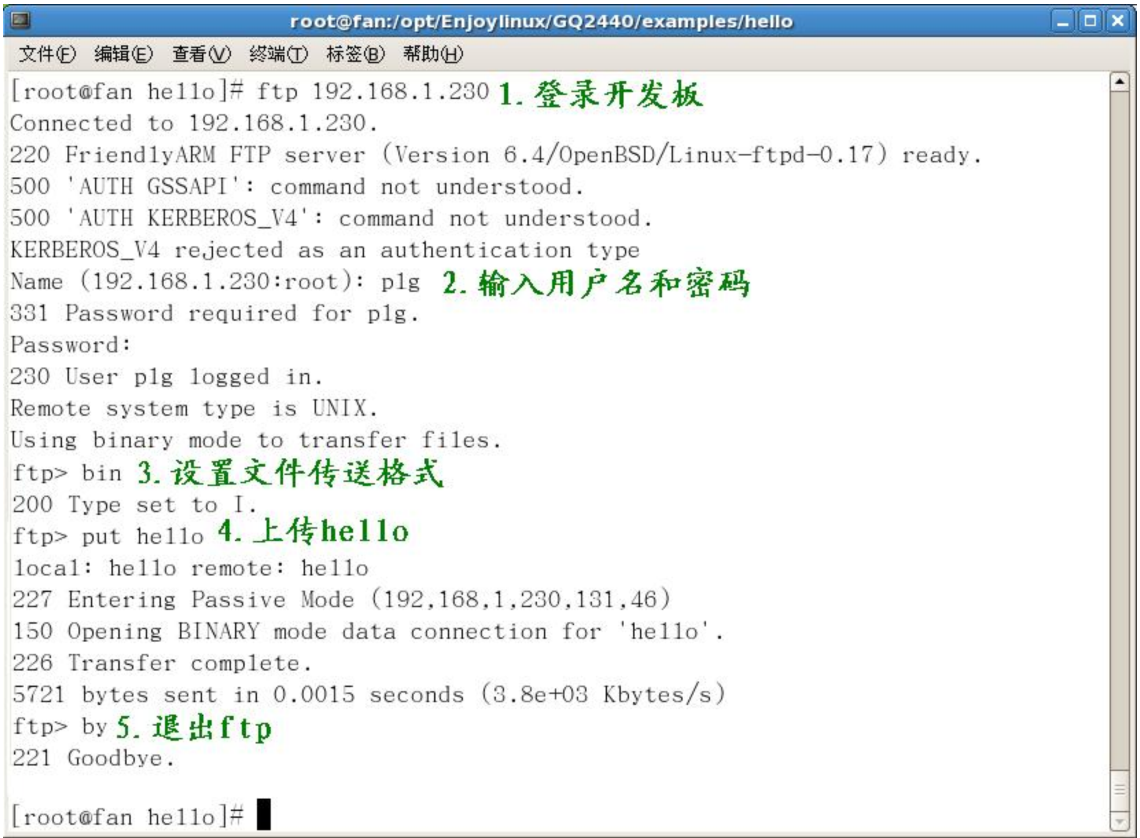
注意：如果此时强制拔出优盘，需要退回到根目录，再执行 **umount /mnt** 方可为下一次做好挂载的准备。



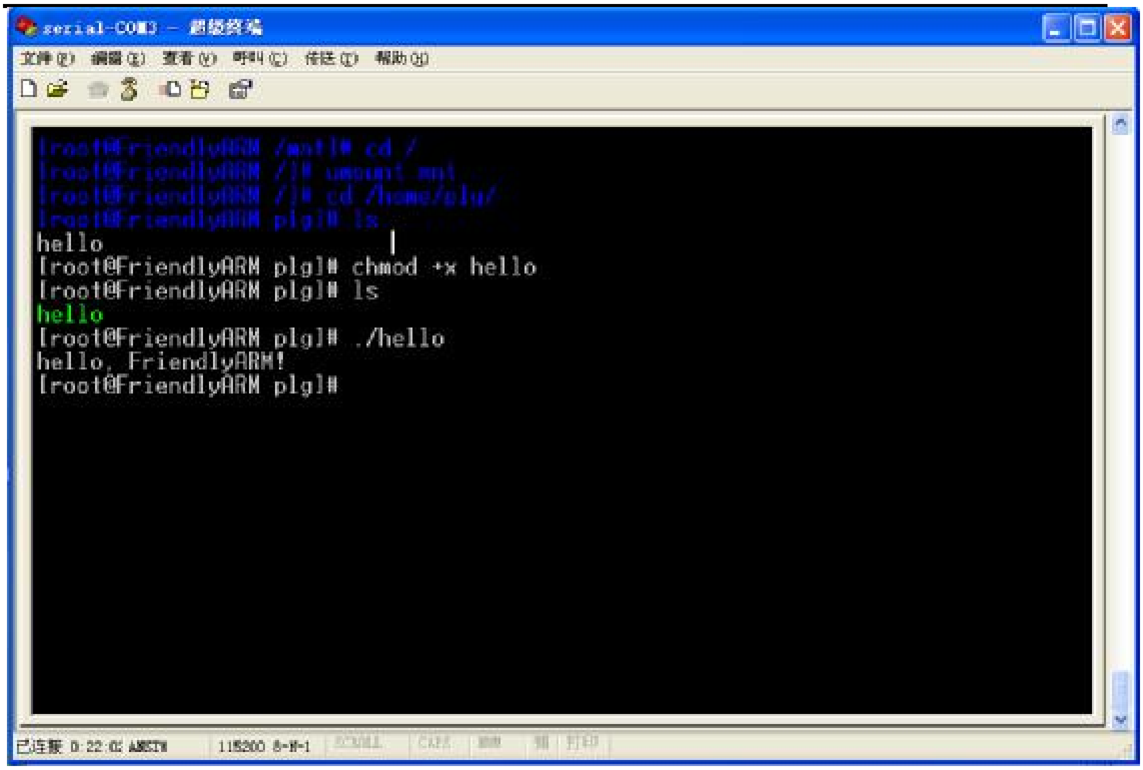
(2)使用 ftp 传送文件(推荐使用)

方法：使用 ftp 登录目标板，把编译好的程序上传；然后修改上传后目标板上的程序的可执行属性，并执行。

首先，在 PC 端执行，如图所示：



然后，在目标板一端执行，如图所示：



(3)通过串口传送文件到开发板

通过 2.5.5 章节我们学会了如何通过串口传送文件到开发板，您也可以通过相同的方法传送 hello 可执行程序，具体步骤在此不再详细描述，记得传送完毕把文件的属性改为可执行才能正常运行。

#chmod +x hello

说明：有些用户使用 USB 转串口线，因为有些转接器性能是不太好的，所以有时会出现"传输超时"或者根本无法传输到开发板的现象，因此我们建议使用 ftp 传送到开发板。

(4)通过网络文件系统 NFS 执行

Linux 中最常用的方法就是采用 NFS 来执行各种程序，这样可以不必花费很多时间下载程序，虽然在此下载 hello 程序用不了多久，一旦您的应用程序变得越来越大，您就会发现使用 NFS 运行的方便所在。

如同前面所讲述的那样，请先按照 5.5 章节搭建好 NFS 服务器系统，然后在命令行

输入以下命令（假定服务器的 IP 地址为 192.168.1.5）：

#mount –t nfs –o nolock 192.168.1.5: /opt/Enjoylinux/GQ2440/root_qtopia /mnt/

挂接成功，您就可以进入/mnt 目录进行操作了，在您的 PC Linux 终端把 hello 复制到**/opt/Enjoylinux/GQ2440/root_qtopia** 目录，然后在开发板的串口终端执行

#cd /mnt

#./hello

7.2 嵌入式 Linux 程序开发入门

7.2.1 LED 测试程序

驱动程序名称：GQ2440_leds.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char
设备类型：misc
设备名：/dev/leds
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/leds
测试程序名称：led.c
测试程序可执行文件名称：led
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

测试程序清单：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
```

```
{
	int on;
	int led_no;
	int fd;
	if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d", &on) != 1 ||
		on < 0 || on > 1 || led_no < 0 || led_no > 3) {
			fprintf(stderr, "Usage: leds led_no 0|1\n");
			exit(1);
	}
	fd = open("/dev/leds0", 0);
	if (fd < 0) {
		fd = open("/dev/leds", 0);
	}
	if (fd < 0) {
		perror("open device leds");
		exit(1);
	}
	ioctl(fd, on, led_no);
	close(fd);
	return 0;
}
```

你可以按照上面的 hello 程序的步骤手工编译出led 可执行文件，然后下载到开发板运行它。

7.2.2 按键测试程序

驱动程序名称：GQ2440_buttons.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char
设备类型：misc
设备名：/dev/buttons
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/buttons

测试程序名称：buttons_test.c

测试程序可执行文件名称：buttons

说明：Button驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

测试程序清单：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
{
	int buttons_fd;
	char buttons[6] = {'0', '0', '0', '0', '0', '0'};

	buttons_fd = open("/dev/buttons", 0);
	if (buttons_fd < 0) {
		perror("open device buttons");
		exit(1);
	}

	for (;;) {
		char current_buttons[6];
		int count_of_changed_key;
		int i;
		if (read(buttons_fd, current_buttons, sizeof current_buttons) != sizeof
```

```
current_buttons) {
                        perror("read buttons:");
                        exit(1);
                }

                for (i = 0, count_of_changed_key = 0; i < sizeof buttons / sizeof buttons[0]; i++) {
                        if (buttons[i] != current_buttons[i]) {
                                buttons[i] = current_buttons[i];
                                printf("%skey %d is %s", count_of_changed_key? ", ": "", i+1, buttons[i] ==
'0' ? "up" : "down");
                                count_of_changed_key++;
                        }
                }
                if (count_of_changed_key) {
                        printf("\n");
                }


        }

        close(buttons_fd);
        return 0;
}
```

7.2.3 PWM 测试程序

驱动程序名称：GQ2440_pwm.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char
设备类型：misc
设备名：/dev/pwm
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/pwm
测试程序名称：pwm_test.c

测试程序可执行文件名称：pwm_test
说明：PWM驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

测试程序清单：

```
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>

#define PWM_IOCTL_SET_FREQ          1
#define PWM_IOCTL_STOP              2

#define   ESC_KEY           0x1b

static int getch(void)
{
        struct termios oldt,newt;
        int ch;

        if (!isatty(STDIN_FILENO)) {
                fprintf(stderr, "this problem should be run at a terminal\n");
                exit(1);
        }
        // save terminal setting
        if(tcgetattr(STDIN_FILENO, &oldt) < 0) {
                perror("save the terminal setting");
                exit(1);
        }

        // set terminal as need
        newt = oldt;
        newt.c_lflag &= ~( ICANON | ECHO );
```

```
	if(tcsetattr(STDIN_FILENO,TCSANOW, &newt) < 0) {
		perror("set terminal");
		exit(1);
	}

	ch = getchar();

	// restore termial setting
	if(tcsetattr(STDIN_FILENO,TCSANOW,&oldt) < 0) {
		perror("restore the termial setting");
		exit(1);
	}
	return ch;
}

static int fd = -1;
static void close_buzzer(void);
static void open_buzzer(void)
{
	fd = open("/dev/pwm", 0);
	if (fd < 0) {
		perror("open pwm_buzzer device");
		exit(1);
	}

	// any function exit call will stop the buzzer
	atexit(close_buzzer);
}

static void close_buzzer(void)
{
	if (fd >= 0) {
```

```
            ioctl(fd, PWM_IOCTL_STOP);
            close(fd);
            fd = -1;
      }
}

static void set_buzzer_freq(int freq)
{
      // this IOCTL command is the key to set frequency
      int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, freq);
      if(ret < 0) {
            perror("set the frequency of the buzzer");
            exit(1);
      }
}
static void stop_buzzer(void)
{
      int ret = ioctl(fd, PWM_IOCTL_STOP);
      if(ret < 0) {
            perror("stop the buzzer");
            exit(1);
      }
}

int main(int argc, char **argv)
{
      int freq = 1000 ;

      open_buzzer();

      printf( "\nBUZZER TEST ( PWM Control )\n" );
      printf( "Press +/- to increase/reduce the frequency of the BUZZER\n" ) ;
```

```
    printf( "Press 'ESC' key to Exit this program\n\n" );


    while( 1 )
    {
        int key;

        set_buzzer_freq(freq);
        printf( "\tFreq = %d\n", freq );

        key = getch();

        switch(key) {
        case '+':
            if( freq < 20000 )
                freq += 10;
            break;

        case '-':
            if( freq > 11 )
                freq -= 10 ;
            break;

        case ESC_KEY:
        case EOF:
            stop_buzzer();
            exit(0);

        default:
            break;
        }
    }
```

```
}
```

7.2.4 I2C-EEPROM 编程示例

驱动程序名称：i2c-s3c2410.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/i2c/busses
设备类型：char
设备名：/dev/i2c/0
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/i2c
测试程序名称：eeprog.c 24cXX.c
测试程序可执行文件名称：i2c
说明：i2c驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

测试程序清单：
注意：以下程序还需同目录下 24cXX.c 程序的支持

```
#include <stdio.h>
#include <fcntl.h>
#include <getopt.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "24cXX.h"

#define usage_if(a) do { do_usage_if( a , __LINE__); } while(0);
void do_usage_if(int b, int line)
{
```

```
	const static char *eeprog_usage =
		"I2C-24C08(256 bytes) Read/Write Program, ONLY FOR TEST!\n"
		"FriendlyARM Computer Tech. 2009\n";
	if(!b)
		return;
	fprintf(stderr, "%s\n[line %d]\n", eeprog_usage, line);
	exit(1);
}


#define die_if(a, msg) do { do_die_if( a , msg, __LINE__); } while(0);
void do_die_if(int b, char* msg, int line)
{
	if(!b)
		return;
	fprintf(stderr, "Error at line %d: %s\n", line, msg);
	fprintf(stderr, "	sysmsg: %s\n", strerror(errno));
	exit(1);
}


static int read_from_eeprom(struct eeprom *e, int addr, int size)
{
	int ch, i;
	for(i = 0; i < size; ++i, ++addr)
	{
		die_if((ch = eeprom_read_byte(e, addr)) < 0, "read error");
		if( (i % 16) == 0 )
			printf("\n %.4x|    ", addr);
		else if( (i % 8) == 0 )
			printf("   ");
		printf("%.2x ", ch);
```

```
			fflush(stdout);
		}
		fprintf(stderr, "\n\n");
		return 0;
}

static int write_to_eeprom(struct eeprom *e, int addr)
{
		int i;
		for(i=0, addr=0; i<256; i++, addr++)
		{
			if( (i % 16) == 0 )
				printf("\n %.4x|    ", addr);
			else if( (i % 8) == 0 )
				printf("    ");
			printf("%.2x ", i);
			fflush(stdout);
			die_if(eeprom_write_byte(e, addr, i), "write error");
		}
		fprintf(stderr, "\n\n");
		return 0;
}

int main(int argc, char** argv)
{
		struct eeprom e;
		int op;

		op = 0;

		usage_if(argc != 2 || argv[1][0] != '-' || argv[1][2] != '\0');
		op = argv[1][1];
```

```
        fprintf(stderr, "Open /dev/i2c/0 with 8bit mode\n");
        die_if(eeprom_open("/dev/i2c/0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e) < 0,
                "unable to open eeprom device file "
                "(check that the file exists and that it's readable)");
        switch(op)
        {
        case 'r':
                fprintf(stderr, "   Reading 256 bytes from 0x0\n");
                read_from_eeprom(&e, 0, 256);
                break;
        case 'w':
                fprintf(stderr, "   Writing 0x00-0xff into 24C08 \n");
                write_to_eeprom(&e, 0);
                break;
        default:
                usage_if(1);
                exit(1);
        }
        eeprom_close(&e);

        return 0;
}
```

7.2.5 串口编程示例

驱动程序名称：s3c2440.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/serial/
设备类型：char
设备名：/dev/ttySAC0,1,2
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/comtest

测试程序名称：comtest.c

测试程序可执行文件名称：armcomtest

说明：测试程序编译后可得到x86 版本和arm 版本，其源代码是完全一样的。

测试程序清单：

```
# include <stdio.h>
# include <stdlib.h>
# include <termio.h>
# include <unistd.h>
# include <fcntl.h>
# include <getopt.h>
# include <time.h>
# include <errno.h>
# include <string.h>

static void Error(const char *Msg)
{
	fprintf (stderr, "%s\n", Msg);
	fprintf (stderr, "strerror() is %s\n", strerror(errno));
	exit(1);
}
static void Warning(const char *Msg)
{
	fprintf (stderr, "Warning: %s\n", Msg);
}



static int SerialSpeed(const char *SpeedString)
{
	int SpeedNumber = atoi(SpeedString);
#	define TestSpeed(Speed) if (SpeedNumber == Speed) return B##Speed
	TestSpeed(1200);
```

```
	TestSpeed(2400);
	TestSpeed(4800);
	TestSpeed(9600);
	TestSpeed(19200);
	TestSpeed(38400);
	TestSpeed(57600);
	TestSpeed(115200);
	TestSpeed(230400);
	Error("Bad speed");
	return -1;
}

static void PrintUsage(void)
{

	fprintf(stderr, "comtest - interactive program of comm port\n");
	fprintf(stderr, "press [ESC] 3 times to quit\n\n");

	fprintf(stderr, "Usage: comtest [-d device] [-t tty] [-s speed] [-7] [-c] [-x] [-o] [-h]\n");
	fprintf(stderr, "         -7 7 bit\n");
	fprintf(stderr, "         -x hex mode\n");
	fprintf(stderr, "         -o output to stdout too\n");
	fprintf(stderr, "         -c stdout output use color\n");
	fprintf(stderr, "         -h print this help\n");
	exit(-1);
}

static inline void WaitFdWriteable(int Fd)
{
	fd_set WriteSetFD;
	FD_ZERO(&WriteSetFD);
	FD_SET(Fd, &WriteSetFD);
```

```
        if (select(Fd + 1, NULL, &WriteSetFD, NULL, NULL) < 0) {
            Error(strerror(errno));
        }

}

int main(int argc, char **argv)
{
    int CommFd, TtyFd;

    struct termios TtyAttr;
    struct termios BackupTtyAttr;

    int DeviceSpeed = B115200;
    int TtySpeed = B115200;
    int ByteBits = CS8;
    const char *DeviceName = "/dev/ttyS0";
    const char *TtyName = "/dev/tty";
    int OutputHex = 0;
    int OutputToStdout = 0;
    int UseColor = 0;

    opterr = 0;
    for (;;) {
        int c = getopt(argc, argv, "d:s:t:7xoch");
        if (c == -1)
            break;
        switch(c) {
        case 'd':
            DeviceName = optarg;
            break;
        case 't':
```

```
			TtyName = optarg;
			break;
		case 's':
			if (optarg[0] == 'd') {
				DeviceSpeed = SerialSpeed(optarg + 1);
			} else if (optarg[0] == 't') {
				TtySpeed = SerialSpeed(optarg + 1);
			} else
				TtySpeed = DeviceSpeed = SerialSpeed(optarg);
			break;
		case 'o':
			OutputToStdout = 1;
			break;
		case '7':
			ByteBits = CS7;
			break;
		case 'x':
			OutputHex = 1;
			break;
		case 'c':
			UseColor = 1;
			break;
		case '?':
		case 'h':
		default:
			PrintUsage();
		}
	}
	if (optind != argc)
		PrintUsage();

	CommFd = open(DeviceName, O_RDWR, 0);
```

```
	if (CommFd < 0)
		Error("Unable to open device");
	if (fcntl(CommFd, F_SETFL, O_NONBLOCK) < 0)
		Error("Unable set to NONBLOCK mode");



	memset(&TtyAttr, 0, sizeof(struct termios));
	TtyAttr.c_iflag = IGNPAR;
	TtyAttr.c_cflag = DeviceSpeed | HUPCL | ByteBits | CREAD | CLOCAL;
	TtyAttr.c_cc[VMIN] = 1;

	if (tcsetattr(CommFd, TCSANOW, &TtyAttr) < 0)
		Warning("Unable to set comm port");

	TtyFd = open(TtyName, O_RDWR | O_NDELAY, 0);
	if (TtyFd < 0)
		Error("Unable to open tty");

	TtyAttr.c_cflag = TtySpeed | HUPCL | ByteBits | CREAD | CLOCAL;
	if (tcgetattr(TtyFd, &BackupTtyAttr) < 0)
		Error("Unable to get tty");

	if (tcsetattr(TtyFd, TCSANOW, &TtyAttr) < 0)
		Error("Unable to set tty");


	for (;;) {
		unsigned char Char = 0;
		fd_set ReadSetFD;

		void OutputStdChar(FILE *File) {
```

```
            char Buffer[10];
            int Len = sprintf(Buffer, OutputHex ? "%.2X  " : "%c", Char);
            fwrite(Buffer, 1, Len, File);
        }

        FD_ZERO(&ReadSetFD);

        FD_SET(CommFd, &ReadSetFD);
        FD_SET( TtyFd, &ReadSetFD);
#    define max(x,y) ( ((x) >= (y)) ? (x) : (y) )
        if (select(max(CommFd, TtyFd) + 1, &ReadSetFD, NULL, NULL, NULL) < 0) {
            Error(strerror(errno));
        }
#    undef max

        if (FD_ISSET(CommFd, &ReadSetFD)) {
            while (read(CommFd, &Char, 1) == 1) {

                WaitFdWriteable(TtyFd);
                if (write(TtyFd, &Char, 1) < 0) {
                    Error(strerror(errno));
                }
                if (OutputToStdout) {
                    if (UseColor)
                        fwrite("\x1b[01;34m", 1, 8, stdout);
                    OutputStdChar(stdout);
                    if (UseColor)
                        fwrite("\x1b[00m", 1, 8, stdout);
                    fflush(stdout);
                }
            }
        }
```

```
            if (FD_ISSET(TtyFd, &ReadSetFD)) {
                while (read(TtyFd, &Char, 1) == 1) {
                    static int EscKeyCount = 0;
                    WaitFdWriteable(CommFd);
                    if (write(CommFd, &Char, 1) < 0) {
                        Error(strerror(errno));
                    }
                    if (OutputToStdout) {
                        if (UseColor)
                            fwrite("\x1b[01;31m", 1, 8, stderr);
                        OutputStdChar(stderr);
                        if (UseColor)
                            fwrite("\x1b[00m", 1, 8, stderr);
                        fflush(stderr);
                    }

                    if (Char == '\x1b') {
                        EscKeyCount ++;
                        if (EscKeyCount >= 3)
                            goto ExitLabel;
                    } else
                        EscKeyCount = 0;
                }
            }

        }

ExitLabel:
    if (tcsetattr(TtyFd, TCSANOW, &BackupTtyAttr) < 0)
        Error("Unable to set tty");
```

```
    return 0;
}
```

7.2.6 UDP 网络编程

驱动程序名称：dm9000.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/net/
设备名：eth0 (网络设备并不在/dev 目录中出现)
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/udptak
测试程序名称：udptak.c
测试程序可执行文件名称：udptak
说明：测试程序编译后可得到x86 版本和arm 版本，其源代码是完全一样的。

TCP/IP 提供了无连接的传输层协议：UDP(User Datagram Protocol，即用户数据报协议)。UDP 与TCP 有很大的区别，因为无连接的socket 编程与面向连接的socket 编程也有很大的差异。由于不用建立连接，因此每个发送个接收的数据报都包含了发送方和接收方的地址信息。

在发送和接收数据之前，先要建立一个数据报方式的套接字，该 socket 的类型为SOCK_DGRAM，用如下的调用产生：

sockfd=socket(AF_INET, SOCK_DGRAM, 0);

由于不需要建立连接，因此产生socket 后就可以直接发送和接收了。当然，要接收数据报也必须绑定一个端口，否则发送方无法得知要发送到哪个端口。Sendto 和recvfrom 两个系统调用分别用于发送和接收数据报，其调用格式为：

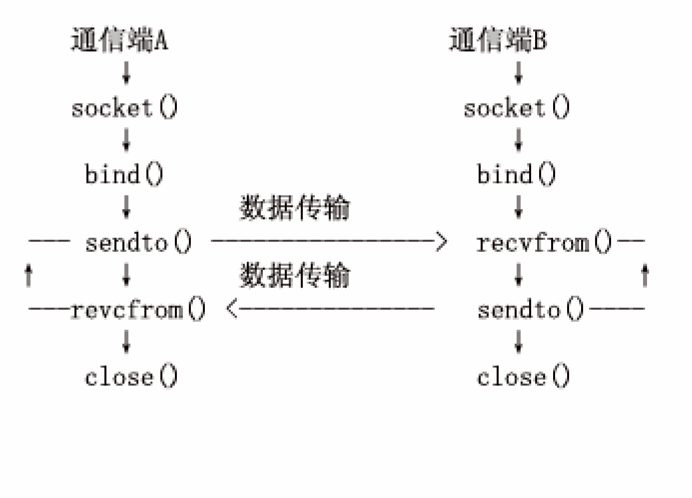
int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, inttolen);

int recvfrom(int, s, void *buf, int len, unsigned int flags, struct sockaddr *from, intfromlen);

其中s 为所使用的socket，msg 和buf 分别为发送和接收的缓冲区指针，len 为缓冲区的长度，flags 为选项标志，此处还用不到，设为0 即可。to 和from 就是发送的目的地址和接收的来源地址，包含了IP 地址和端口信息。tolen 和fromlen 分别是to 和from 这

两个socket地址结构的长度。这两个函数的返回值就是实际发送和接收的字节数，返回-1表示出错。

使用无连接方式通信的基本过程如图所示。



UDP 通信的基本过程

上图描述的是通信双方都绑定自己地址端口的情形，但在某些情况下，也可能有一方不用绑定地址和端口。不绑定的一方的地址和端口由内核分配。由于对方无法预先知道不绑定的一方的端口和IP 地址(假设主机有多个端口，这些端口分配了不同的IP 地址)，因此只能由不绑定的一方先发出数据报，对方根据收到的数据报中的来源地址就可以确定回送数据报所需要的发送地址了。显然，在这种情况下对方必须绑定地址和端口，并且通信只能由非绑定方发起。

与 read()和write()相似，进程阻塞在recvfrom()和sendto()中也会发生。但与TCP 方式不同的是，接收到一个字节数为0 的数据报是有可能的，应用程序完全可以将sendto()中的msg 设为NULL，同时将len 设为0。

下面是一个基于以上原理分析的一个 udp 编程的例子:

测试程序清单:

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <stdio.h>

#define BUFLEN 255

int main(int argc, char **argv)
{
	struct sockaddr_in peeraddr, localaddr;
	int sockfd;
	char recmsg[BUFLEN+1];
	int socklen, n;

	if(argc!=5){
		printf("%s <dest IP address> <dest port> <source IP address> <source port>\n",
argv[0]);
		exit(0);
	}

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);
	if(sockfd<0){
		printf("socket creating err in udptalk\n");
		exit(1);
	}
	socklen = sizeof(struct sockaddr_in);
	memset(&peeraddr, 0, socklen);
	peeraddr.sin_family=AF_INET;
	peeraddr.sin_port=htons(atoi(argv[2]));
	if(inet_pton(AF_INET, argv[1], &peeraddr.sin_addr)<=0){
		printf("Wrong dest IP address!\n");
		exit(0);
	}
	memset(&localaddr, 0, socklen);
```

```
    localaddr.sin_family=AF_INET;
    if(inet_pton(AF_INET, argv[3], &localaddr.sin_addr)<=0){
        printf("Wrong source IP address!\n");
        exit(0);
    }
    localaddr.sin_port=htons(atoi(argv[4]));
    if(bind(sockfd, &localaddr, socklen)<0){
        printf("bind local address err in udptalk!\n");
        exit(2);
    }

    if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
    if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
        printf("sendto err in udptalk!\n");
        exit(3);
    }

    for(;;){
        /*recv&send message loop*/
        n = recvfrom(sockfd, recmsg, BUFLEN, 0, &peeraddr, &socklen);
        if(n<0){
            printf("recvfrom err in udptalk!\n");
            exit(4);
        }else{
            recmsg[n]=0;
            printf("peer:%s", recmsg);
        }
        if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
        if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
            printf("sendto err in udptalk!\n");
            exit(3);
        }
```

```
    }
}
```

将 udptalk.c 编译好后就可以运行了，/opt/Enjoylinux/GQ2440/examples/udptalk 目录下的Makefile 指定了两个编译目标可执行文件，一个用于在主机端的x86-udptalk，一个是用于开发板的arm-udptalk，运行make 命令将把这两个程序一起编译出来。可以把arm-udptalk使用上面介绍的方法下载到开发板中(预装的Linux 不含该程序)，假设主机的IP 地址为192.168.1.5，开发板的IP 地址为192.168.1.230。

在主机的终端上输入:

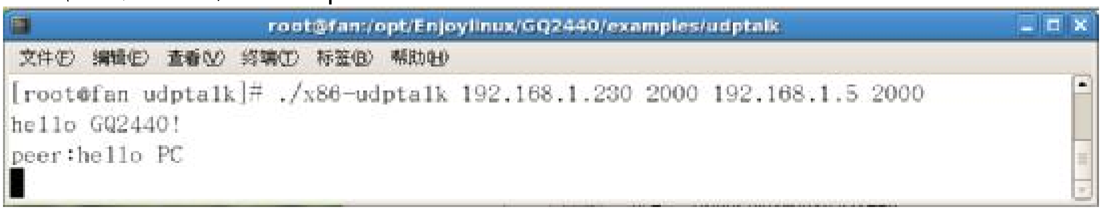
#./x86-udptalk 192.168.1.230 2000 192.168.1.5 2000

在开发板上的终端输入

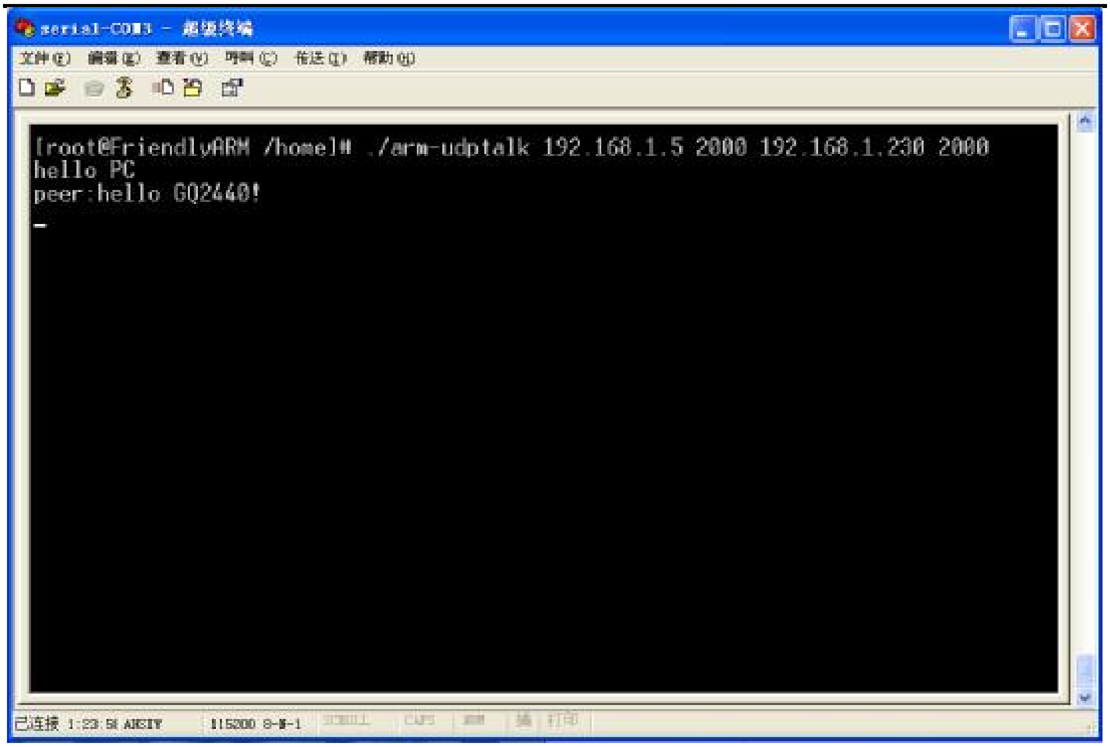
#./arm-udptalk 192.168.1.5 2000 192.168.1.230 2000

则运行结果分别如图所示:

在主机上运行 x86-udptalk:



开发板上运行arm-udptalk:



7.2.7 数学函数库调用示例

测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/math
测试程序名称：mathtest.c
测试程序可执行文件名称：mathtest
说明：使用数学函数的关键是要包含其头文件math.h，并且在编译的时候加入数学函数库libm。

测试程序清单：

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>

int main(void)
{
	double a=8.733243;

	printf("sqrt(%f)=%f\n", a, sqrt(a));

	return 0;
}
```

相应的 Makefile 内容:

```
CROSS=arm-linuxall:
mathtest
#注意：该处包含了数学函数库libm,
mathtest:
	$(CROSS)gcc -o mathtest main.c -lm
clean:
	@rm -vf mathtest *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出mathtest 可执行文件，然后下载到开发板运行它。

7.2.8 线程编程示例

测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/pthread
测试程序名称：pthread_test.c
测试程序可执行文件名称：pthread_test
说明：使用线程的关键是要包含其头文件pthread.h，并且在编译的时候加入线程库libpthread。

测试程序清单:

```
#include<stddef.h>
#include<stdio.h>
#include<unistd.h>
#include"pthread.h"

void reader_function(void);
void writer_function(void);
char buffer;
int buffer_has_item=0;
pthread_mutex_t mutex;
main()
{
	pthread_t reader;
	pthread_mutex_init(&mutex,NULL);
	pthread_create(&reader,NULL,(void*)&reader_function,NULL);
	writer_function();
}
void writer_function(void)
{
	while(1)
	{
		pthread_mutex_lock(&mutex);
		if(buffer_has_item==0)
		{
			buffer='a';
			printf("make a new item\n");
			buffer_has_item=1;
		}
		pthread_mutex_unlock(&mutex);
```

```
    }
}
void reader_function(void)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        if(buffer_has_item==1)
        {
            buffer='\0';
            printf("consume  item\n");
            buffer_has_item=0;
        }
        pthread_mutex_unlock(&mutex);
    }
}
```

相应的 Makefile 内容:

```
CROSS=arm-linux-

all: pthread_test

pthread_test:
	$(CROSS)gcc -static -o pthread_test pthread_test.c -lpthread

clean:
	@rm -vf pthread_test *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出pthread可执行文件，然后下载到开发板运行它。

7.2.9 管道应用编程示例-网页控制 LED

测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/led-player
测试程序名称：led-player.c
测试程序可执行文件名称：led-player

开机后我们可以通过网页发送命令控制开发板上的 LED 闪烁模式，其实这是进程间通信共享资源的一个典型例子，进程间通信就是IPC(InterProcess Communication)，进程间通信的目的一般有：

(1)数据传输
(2)共享数据
(3)通知事件
(4)资源共享
(5)进程控制.

Linux 支持多种IPC 机制，信号和管道是其中的两个。关于更详细的进程间通信的介绍，一般Linux 编程的书上都有介绍，在此我们不多说了。

通过网页来控制 LED 的闪烁模式就是通过管道机制来实现的，其中LED 是共享资源，
led-player 是一个后台程序，它启动的时候就创建了一个命名管道/tmp/ led-control(当然该管道也可以通过命令mknod 来创建，那样程序就要改写了，有兴趣的可以自己试试)， 并一直监测输入该管道的数据，根据不同的参数(模式type 和周期period)来改变LED 的显示模式；leds.cgi 是一个网关程序，它接收从网页发送过来的字符形式指令（ping 代表跑马灯模式或者乒乓模式，counter 代表计数器模式，stop 代表停止模式，slow 代表周期为0.25m，normal 代表周期为0.125m，fast 代表周期为0.0625m），并对这些指令进行赋值转换为实际数字，然后调用echo 命令输送到管道/tmp/ led-control 以此实现对LED 的控制，以下是各自的程序清单。

测试程序清单：

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <string.h>

static int led_fd;
static int type = 1;

static void push_leds(void)
{
	static unsigned step;
	unsigned led_bitmap;
	int i;

	switch(type) {
	case 0:
		if (step >= 6) {
			step = 0;
		}
		if (step < 3) {
			led_bitmap = 1 << step;
		} else {
			led_bitmap = 1 << (6 - step);
		}
		break;
	case 1:
		if (step > 255) {
			step = 0;
```

```
            }
            led_bitmap = step;
            break;
        default:
            led_bitmap = 0;
        }
        step++;
        for (i = 0; i < 4; i++) {
            ioctl(led_fd, led_bitmap & 1, i);
            led_bitmap >>= 1;
        }
}

int main(void)
{
        int led_control_pipe;
        int null_writer_fd; // for read endpoint not blocking when control process exit

        double period = 0.5;

        led_fd = open("/dev/leds0", 0);
        if (led_fd < 0) {
            led_fd = open("/dev/leds", 0);
        }
        if (led_fd < 0) {
            perror("open device leds");
            exit(1);
        }
        unlink("/tmp/led-control");
        mkfifo("/tmp/led-control", 0666);

        led_control_pipe = open("/tmp/led-control", O_RDONLY | O_NONBLOCK);
```

```
    if (led_control_pipe < 0) {
        perror("open control pipe for read");
        exit(1);
    }
    null_writer_fd = open("/tmp/led-control", O_WRONLY | O_NONBLOCK);
    if (null_writer_fd < 0) {
        perror("open control pipe for write");
        exit(1);
    }

    for (;;) {
        fd_set rds;
        struct timeval step;
        int ret;

        FD_ZERO(&rds);
        FD_SET(led_control_pipe, &rds);
        step.tv_sec    = period;
        step.tv_usec = (period - step.tv_sec) * 1000000L;

        ret = select(led_control_pipe + 1, &rds, NULL, NULL, &step);
        if (ret < 0) {
            perror("select");
            exit(1);
        }
        if (ret == 0) {
            push_leds();
        } else if (FD_ISSET(led_control_pipe, &rds)) {
            static char buffer[200];
            for (;;) {
                char c;
                int len = strlen(buffer);
```

```
                        if (len >= sizeof buffer - 1) {
                                memset(buffer, 0, sizeof buffer);
                                break;
                        }
                        if (read(led_control_pipe, &c, 1) != 1) {
                                break;
                        }
                        if (c == '\r') {
                                continue;
                        }
                        if (c == '\n') {
                                int tmp_type;
                                double tmp_period;
                                if (sscanf(buffer,"%d%lf", &tmp_type, &tmp_period) == 2) {
                                        type = tmp_type;
                                        period = tmp_period;
                                }
                                fprintf(stderr, "type is %d, period is %lf\n", type, period);
                                memset(buffer, 0, sizeof buffer);
                                break;
                        }
                        buffer[len] = c;
                }
        }
    }

    close(led_fd);
    return 0;
}
```

使用 make 指令可以直接编译出led-player 可执行文件，它被作为一个服务器放置在

开发板的/sbin 目录中。

Leds.cgi 网关程序源代码(该程序在开发板上的位置：/www/leds.cgi)，可见该网关程序其实就是一个shell 脚本，它被网页leds.html 调用为一个执行“action”：

```
#!/bin/sh

type=0
period=1

case $QUERY_STRING in
*ping*)
type=0
;;
*counter*)
type=1
;;
*stop*)
type=2
;;
esac

case $QUERY_STRING in
*slow*)
period=0.25
;;
*normal*)
period=0.125
;;
*fast*)
period=0.0625
;;
esac
```

```
/bin/echo $type $period > /tmp/led-control

echo "Content-type: text/html; charset=gb2312"
echo
/bin/cat led-result.template

exit 0
```

你可以按照上面的 hello 程序的步骤手工编译出pthread可执行文件，然后下载到开发板运行它。

7.2.10 基于 C++的 Hello,World

测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/c++
测试程序名称：cplus.c++
测试程序可执行文件名称：cplus

测试程序清单：

```
#include <iostream>
#include <cstring>
using namespace std;

class String
{
private:
	char *str;
public:
	String(char *s)
	{
		int lenght=strlen(s);
		str = new char[lenght+1];
		strcpy(str, s);
```

```
    }
    ~String()
    {
        cout << "Deleting str.\n";
        delete[] str;
    }
    void display()
    {
        cout << str <<endl;
    }
};

int main(void)
{
    String s1="I like FriendlyARM.";
    cout << "s1=";
    s1.display();
    return 0;
    double num, ans;

    cout << "Enter num:";
}
```

你可以按照上面的 hello 程序的步骤手工编译出cplus 可执行文件，然后下载到开发板运行它。

7.3 最简单的嵌入式 Linux 驱动程序模块

7.1 一节我们介绍了一个简单的Linux 程序Hello,World，它是运行于用户态的应用程

序，现在我们再介绍一个运行于内核态的Hello，World 程序，它其实是一个最简单的驱动程序模块。

7.3.1 Hello,Module 源代码

驱动程序源代码目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char
驱动程序名称：GQ2440_hello_module.c
说明：该驱动装载后不会在dev 下创建任何设备节点。

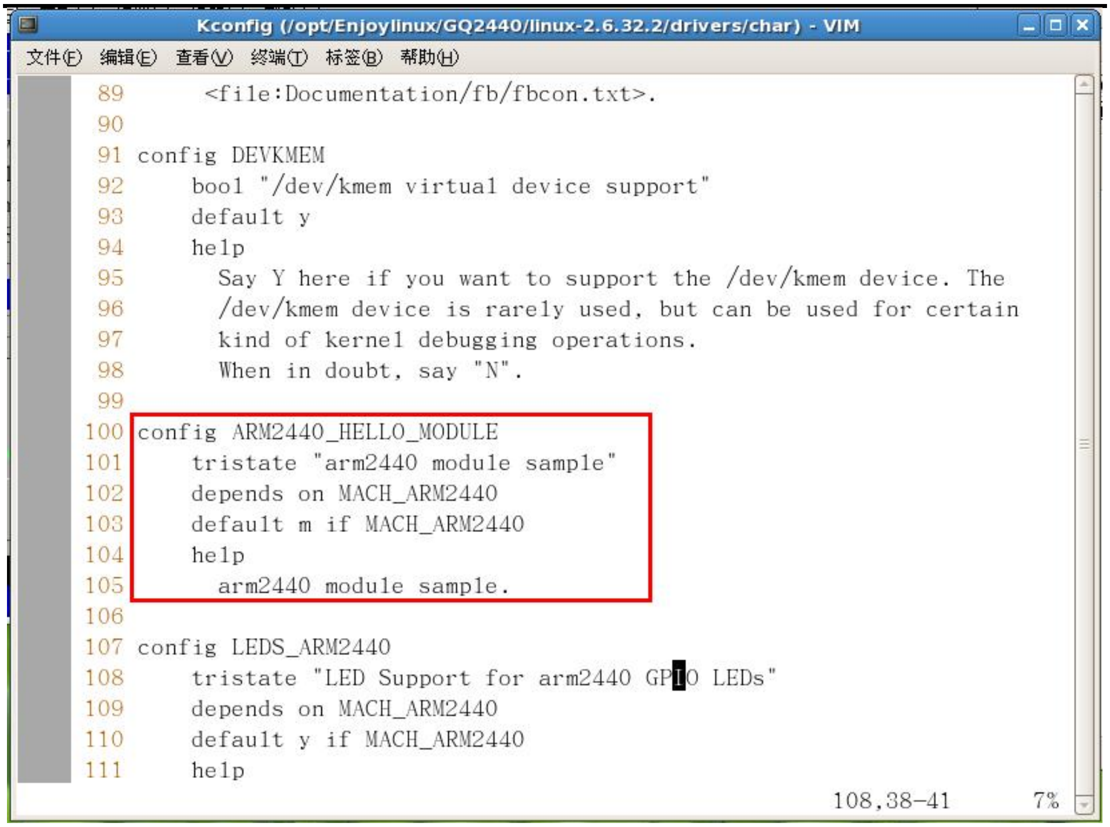
驱动程序清单：

```
#include <linux/kernel.h>
#include <linux/module.h>
static int __init GQ2440_hello_module_init(void)
{
	printk("Hello, GQ2440module is installed !\n");
	return 0;
}
static void __exit GQ2440_hello_module_cleanup(void)
{
	printk("Good-bye, GQ2440module was removed!\n");
}
module_init(GQ2440_hello_module_init);
module_exit(GQ2440_hello_module_cleanup);
MODULE_LICENSE("GPL");
```

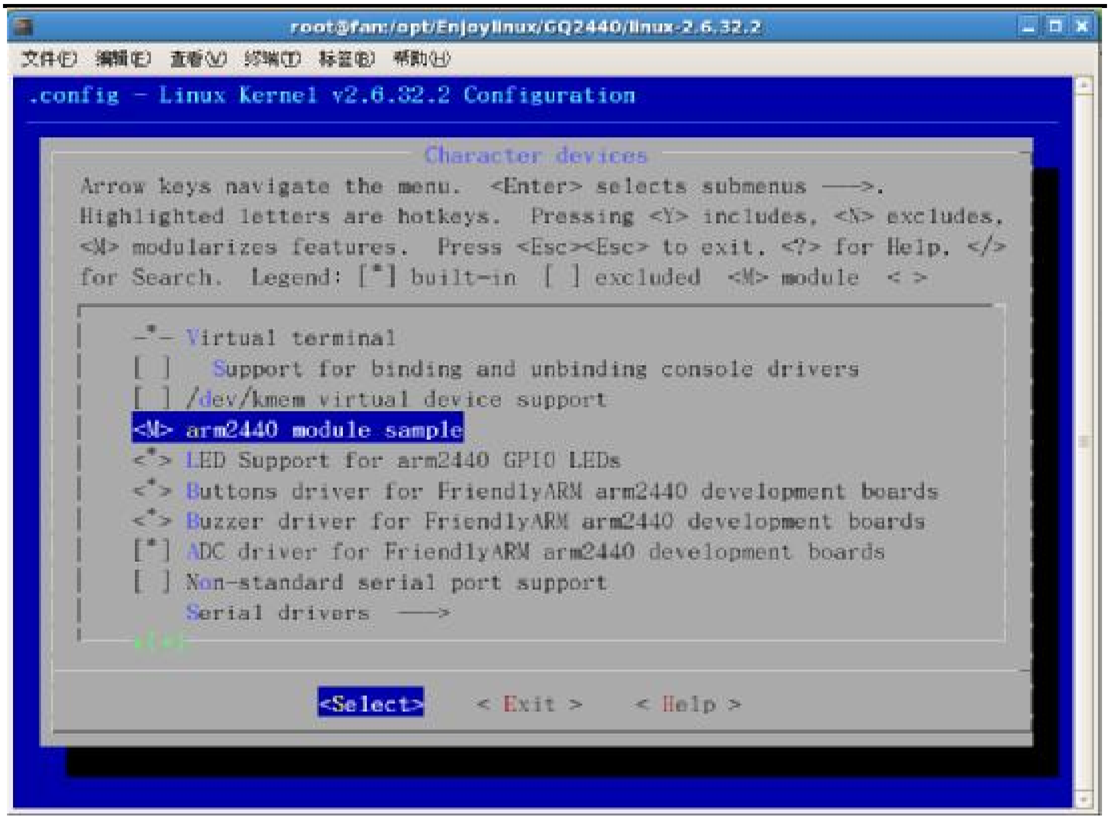
7.3.2 把 Hello,Module 加入内核代码树，并编译

一般编译 2.6 版本的驱动模块需要把驱动代码加入内核代码树，并做相应的配置，如下步骤(注意：实际上以下步骤均已经做好，你只需要打开检查一下直接编译就可以了):

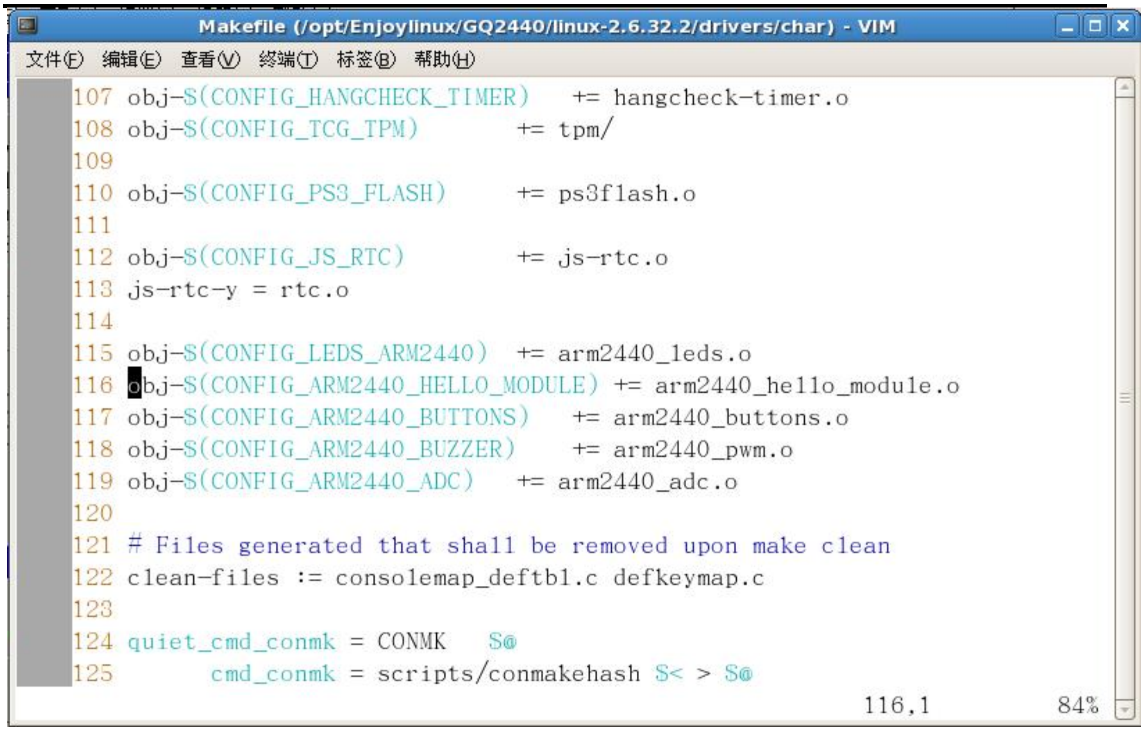
Step1: 编辑配置文件Kconfig，加入驱动选项，使之在make menuconfig 的时候出现
打开 linux-2.6.32.2/drivers/char/Kconfig 文件，添加如图所示：



保存退出，这时在linux-2.6.32.2 目录位置运行一下make menuconfig 就可以在【Device Drivers】|【Character devices】菜单中看到刚才所添加的选项了，按下空格键将会选择为<M>，此意为要把该选项编译为模块方式；再按下空格会变为<*>，意为要把该选项编译到内核中，在此我们选择<M>，如图：



Step2: 通过上一步，我们虽然可以在配置内核的时候进行选择，但实际上此时执行编译内核还是不能把GQ2440_hello_module.c 编译进去的，还需要在Makefile 中把内核配置选项和真正的源代码联系起来，打开 linux-2.6.32.2/drivers/char/Makefile，如图添加并保存退出：



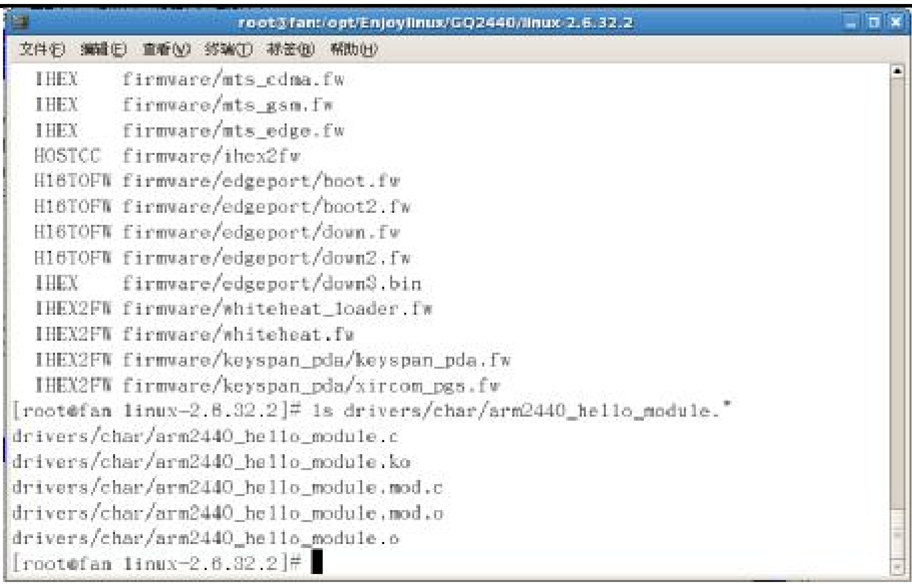
Makefile (/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char) - VIM

文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)

```
107 obj-$(CONFIG_HANGCHECK_TIMER)    += hangcheck-timer.o
108 obj-$(CONFIG_TCG_TPM)          += tpm/
109
110 obj-$(CONFIG_PS3_FLASH)        += ps3flash.o
111
112 obj-$(CONFIG_JS_RTC)           += js-rtc.o
113 js-rtc-y = rtc.o
114
115 obj-$(CONFIG_LEDS_ARM2440)   += arm2440_leds.o
116 obj-$(CONFIG_ARM2440_HELLO_MODULE) += arm2440_hello_module.o
117 obj-$(CONFIG_ARM2440_BUTTONS)    += arm2440_buttons.o
118 obj-$(CONFIG_ARM2440_BUZZER)     += arm2440_pwm.o
119 obj-$(CONFIG_ARM2440_ADC)    += arm2440_adc.o
120
121 # Files generated that shall be removed upon make clean
122 clean-files := consolemap_deftbl.c defkeymap.c
123
124 quiet_cmd_conmk = CONMK    $@
125       cmd_conmk = scripts/conmakehash $< > $@
                                                   116,1          84%
```

Step3: 这时回到linux-2.6.32.2 源代码根目录位置，执行make modules，就可以生成我们所需要的内核模块文件GQ2440_hello_module.ko 了，如图:

至此，我们已经完成了模块驱动的编译。



7.3.3 把 Hello, Module 下载到开发板并安装使用

在此使用 rz 命令把编译出的**GQ2440_hello_module.ko** 下载到板子中，并把它移动到**/lib/modules/2.6.32.2-FriendlyARM** 目录然后在板子中现在执行

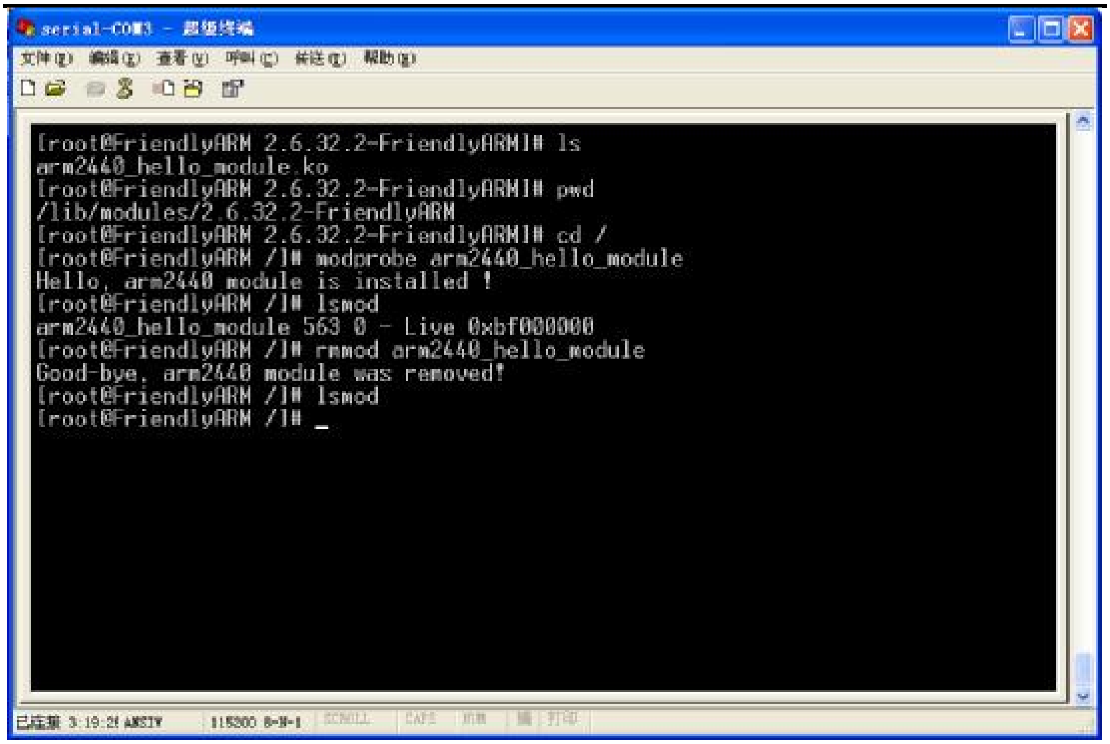
#modprobe GQ2440_hello_module

可以看到该模块已经被装载了(注意：使用modprobe 命令加载模块不需要加“ko”尾缀)再执行以下命令，可以看到该模块被卸载

#rmmod GQ2440_hello_module

注意：要能够正常卸载模块，必须把模块放入开发板的/lib/modules/2.6.32.2-Friendly ARM 目录。

另外需要注意的是：因为我们的内核有时会升级更新，如果内核版本已经改变，请依照具体的内核版本重新建立一个模块存放目录，在此为/lib/modules/2.6.29.4-FriendlyARM 整个过程如下图：



7.4 简易 Linux 驱动程序示例

在上一小节，我们介绍了最简单的Hello,Module 驱动程序模块，它只是从串口输出一些信息，并未对应板上的硬件进行操作，下面的几个例子都是和硬件密切相关的实际驱动，在嵌入式 Linux 系统中，大部分的硬件都需要类似的驱动才能操作，比如触摸屏、网卡、音频等，在这里我们介绍的是一些简单典型的例子，实际上复杂的驱动都有参考代码，不必从头写驱动。

在本节中，我们不介绍驱动程序理论概念之类的内容，那些在网上或者书籍中都有比较系统的描述。

7.4.1 LED 驱动程序

驱动程序名称：GQ2440_leds.c
驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char
设备号：Led 属于 misc 设备，设备自动生成
设备名：/dev/leds
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/leds
测试程序名称：led.c
测试程序可执行文件名称：led
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

要写实际的驱动，就必须了解相关的硬件资源，比如用到的寄存器，物理地址，中断等，在这里，LED 是一个很简单的例子，它用到了如下硬件资源。

开发板上所用到的 4 个LED 的硬件资源

GPB5--LED1
GPB6--LED2
GPB7--LED3
GPB8--LED4

要操作所用到的 IO 口，就要设置它们所用到的寄存器，我们需要调用一些现成的函数或者宏，例如：s3c2410_gpio_cfgpin

为什么是 S3C2410 的呢？因为三星出品的S3C2440 芯片所用的寄存器名称以及资源分配大部分和S3C2410 是相同的，在目前各个版本的Linux 系统中，也大都采用了相同的函数定义和宏定义。

它们从哪里定义？细心的用户或许很快就想到它们和体系结构有关，因此你可以在linux-2.6.32.2/arch/arm/mach-s3c2410/include/mach/hardware.h 文件中找到该函数的定义，关于该函数的实际实现则可以在linux-2.6.32.2/arch/arm/plat-s3c24xx/ gpio.c 中找到，它的内容如下：

```
void s3c2410_gpio_cfgpin(unsigned int pin, unsigned int function)
{
    void __iomem *base = S3C24XX_GPIO_BASE(pin);
```

```
unsigned long mask;
unsigned long con;
unsigned long flags;

if (pin < S3C2410_GPIO_BANKB) {
        mask = 1 << S3C2410_GPIO_OFFSET(pin);
} else {
        mask = 3 << S3C2410_GPIO_OFFSET(pin)*2;
}

switch (function) {
case S3C2410_GPIO_LEAVE:
        mask = 0;
        function = 0;
        break;

case S3C2410_GPIO_INPUT:
case S3C2410_GPIO_OUTPUT:
case S3C2410_GPIO_SFN2:
case S3C2410_GPIO_SFN3:
        if (pin < S3C2410_GPIO_BANKB) {
                function -= 1;
                function &= 1;
                function <<= S3C2410_GPIO_OFFSET(pin);
        } else {
                function &= 3;
                function <<= S3C2410_GPIO_OFFSET(pin)*2;
        }
}

/* modify the specified register wwith IRQs off */
```

```
        local_irq_save(flags);

        con   = __raw_readl(base + 0x00);
        con &= ~mask;
        con |= function;

        __raw_writel(con, base + 0x00);

        local_irq_restore(flags);
}
```

实际上，我们并不需要关心这些，写驱动时只要会使用他们就可以了，除非你所使用的CPU 体系平台尚没有被Linux 所支持，因为大部分常见的嵌入式平台都已经有了很完善的类似定义，你不需要自己去编写。

在下面的驱动程序清单中，你可以看到 s3c2410_gpio_cfgpin 被调用的情况。除此之外，你还需要调用一些和设备驱动密切相关的基本函数，如注册设备misc_register，填写驱动函数结构file_operations，以及像Hello,Module 中那样的module_init 和module_exit 函数等。

有些函数并不是必须的，随着你对 Linux 驱动开发的进一步了解和阅读更多的代码，你自然明白。下面是我们为LED 编写的驱动代码清单：

驱动程序清单：

```
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <asm/irq.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
```

```
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/pci.h>
#include <linux/gpio.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <asm/unistd.h>


#define DEVICE_NAME "leds"

static unsigned long led_table [] = {
	S3C2410_GPB(5),
	S3C2410_GPB(6),
	S3C2410_GPB(7),
	S3C2410_GPB(8),
};

static unsigned int led_cfg_table [] = {
	S3C2410_GPIO_OUTPUT,
	S3C2410_GPIO_OUTPUT,
	S3C2410_GPIO_OUTPUT,
	S3C2410_GPIO_OUTPUT,
};
```

```
static int sbc2440_leds_ioctl(
struct inode *inode,
struct file *file,
	unsigned int cmd,
	unsigned long arg)
{
	switch(cmd) {
case 0:
case 1:
	if (arg > 4) {
		return -EINVAL;
	}
	s3c2410_gpio_setpin(led_table[arg], !cmd);
	return 0;
default:
	return -EINVAL;
	}
}

static struct file_operations dev_fops = {
	.owner	=	THIS_MODULE,
	.ioctl=	sbc2440_leds_ioctl,
};

static struct miscdevice misc = {
	.minor = MISC_DYNAMIC_MINOR,
	.name = DEVICE_NAME,
	.fops = &dev_fops,
};

static int __init dev_init(void)
{
```

```
    int ret;

    int i;

    for (i = 0; i < 4; i++) {
        s3c2410_gpio_cfgpin(led_table[i], led_cfg_table[i]);
        s3c2410_gpio_setpin(led_table[i], 0);
    }

    ret = misc_register(&misc);

    printk (DEVICE_NAME"\tinitialized\n");

    return ret;
}

static void __exit dev_exit(void)
{
    misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Enjoylinux Inc.");
```

7.4.2 按键驱动程序

驱动程序名称：GQ2440_buttons.c

驱动源代码所在目录：/opt/Enjoylinux/GQ2440/linux-2.6.32.2/drivers/char

设备号：misc 设备，设备自动生成
设备名：/dev/buttons
测试程序源代码目录：/opt/Enjoylinux/GQ2440/examples/buttons
测试程序名称：buttons_test.c
测试程序可执行文件名称：buttons
说明：按键驱动已经被编译到缺省内核中，因此不能再使用insmod 方式加载。

开发板所用到的按键资源如下：

	K1	K2	K3	K4	K5	K6
中断	EINT8	EINT11	EINT13	EINT14	EINT15	EINT19
对应 GPIO	GPG0	GPG3	GPG5	GPG6	GPG7	GPG11

驱动程序清单：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <asm/irq.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>
#include <linux/sched.h>
#include <linux/gpio.h>
```

```
#define DEVICE_NAME      "buttons"

struct button_irq_desc {
    int irq;
    int pin;
    int pin_setting;
    int number;
    char *name;
};

static struct button_irq_desc button_irqs [] = {
    {IRQ_EINT8 , S3C2410_GPG(0) ,   S3C2410_GPG0_EINT8   , 0, "KEY0"},
    {IRQ_EINT11, S3C2410_GPG(3) ,   S3C2410_GPG3_EINT11 , 1, "KEY1"},
    {IRQ_EINT13, S3C2410_GPG(5) ,   S3C2410_GPG5_EINT13 , 2, "KEY2"},
    {IRQ_EINT14, S3C2410_GPG(6) ,   S3C2410_GPG6_EINT14 , 3, "KEY3"},
    {IRQ_EINT15, S3C2410_GPG(7) ,   S3C2410_GPG7_EINT15 , 4, "KEY4"},
    {IRQ_EINT19, S3C2410_GPG(11),   S3C2410_GPG11_EINT19, 5, "KEY5"},
};
static volatile char key_values [] = {'0', '0', '0', '0', '0', '0'};

static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

static volatile int ev_press = 0;


static irqreturn_t buttons_interrupt(int irq, void *dev_id)
{
    struct button_irq_desc *button_irqs = (struct button_irq_desc *)dev_id;
    int down;

    // udelay(0);
    down = !s3c2410_gpio_getpin(button_irqs->pin);
```

```
	if (down != (key_values[button_irqs->number] & 1)) { // Changed

		key_values[button_irqs->number] = '0' + down;

		ev_press = 1;
		wake_up_interruptible(&button_waitq);
	}

	return IRQ_RETVAL(IRQ_HANDLED);
}


static int s3c24xx_buttons_open(struct inode *inode, struct file *file)
{
	int i;
	int err = 0;

	for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
		if (button_irqs[i].irq < 0) {
			continue;
		}
		err = request_irq(button_irqs[i].irq, buttons_interrupt, IRQ_TYPE_EDGE_BOTH,
			button_irqs[i].name, (void *)&button_irqs[i]);
		if (err)
			break;
	}

	if (err) {
		i--;
		for (; i >= 0; i--) {
			if (button_irqs[i].irq < 0) {
```

```
                        continue;
                    }
                    disable_irq(button_irqs[i].irq);
                    free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
                }
                return -EBUSY;
        }

        ev_press = 1;

        return 0;
}


static int s3c24xx_buttons_close(struct inode *inode, struct file *file)
{
        int i;

        for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
                if (button_irqs[i].irq < 0) {
                        continue;
                }
                free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
        }

        return 0;
}


static int s3c24xx_buttons_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
{
        unsigned long err;
```

```
	if (!ev_press) {
		if (filp->f_flags & O_NONBLOCK)
			return -EAGAIN;
		else
			wait_event_interruptible(button_waitq, ev_press);
	}

	ev_press = 0;

	err = copy_to_user(buff, (const void *)key_values, min(sizeof(key_values), count));

	return err ? -EFAULT : min(sizeof(key_values), count);
}

static unsigned int s3c24xx_buttons_poll( struct file *file, struct poll_table_struct *wait)
{
	unsigned int mask = 0;
	poll_wait(file, &button_waitq, wait);
	if (ev_press)
		mask |= POLLIN | POLLRDNORM;
	return mask;
}


static struct file_operations dev_fops = {
	.owner   =   THIS_MODULE,
	.open    =   s3c24xx_buttons_open,
	.release =   s3c24xx_buttons_close,
	.read    =   s3c24xx_buttons_read,
	.poll    =   s3c24xx_buttons_poll,
};
```

```
static struct miscdevice misc = {
	.minor = MISC_DYNAMIC_MINOR,
	.name = DEVICE_NAME,
	.fops = &dev_fops,
};

static int __init dev_init(void)
{
	int ret;

	ret = misc_register(&misc);

	printk (DEVICE_NAME"\tinitialized\n");

	return ret;
}

static void __exit dev_exit(void)
{
	misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Enjoylinux Inc.");
```

第8章 常见 bootloader 的配置和编译

在 S3C2440/2410 系统中，常见的 bootloader 一般有

- **Vivi** – 由三星提供，韩国 mizi 公司原创，开放源代码，必须使用 arm-linux-gcc 进行编译，目前已经基本停止发展，主要适用于三星 S3C24xx 系列 ARM 芯片，用以启动 Linux 系统，支持串口下载和网络文件系统启动等常用简易功能。
- **armvivi** – 由国嵌提供并积极维护，它基于 vivi 发展而来，不提供源代码，在保留原始 vivi 功能的基础上，整合了诸多其他实用功能，如支持 CRAMFS，YAFFS 文件系统，USB 下载，自动识别并启动 Linux，WinCE, uCos，Vxwork 等多种嵌入式操作系统，下载程序到内存中执行，是 2440/2410 系统中一款功能强大的 bootloader。
- **vboot** – 是一款精简的 bootloader，它的功能很简单，只是启动 Linux 系统，vboot 可以自动适应支持 64M/128M Nand Flash 的 GQ2440 板。
- **U-Boot** – 一个开源的专门针对嵌入式 Linux 系统设计的最流行 bootloader，必须使用 arm-linux-gcc 进行编译，具有强大的网络功能(失去网络，U-Boot 基本丧失其最独到的优势,在 2440/2410 系统中网络是添加网络芯片外扩的,毫无疑问会增加成本,而 USB 是内置的，只需几个电阻配置)，支持网络下载内核并通过网络启动系统，U-Boot 处于更加活跃的更新发展之中，但对于 2440/2410 系统来说，它尚未支持 Nand Flash 启动，国内已经有人为此自行加入了这些功能，本章节中的 U-Boot 即是如此。

Bootloader 以其本身的含义来讲就是下载和启动系统，它类似于 PC 中的 BIOS，大部分芯片厂商所提供的嵌入式系统都提供有这样的程序，而且都比较成熟，大可不必自行编写。

我们所改进的 supervivi 目标是使之更加人性化，更加适合于批量生产需要。下面我们使用图解的方式介绍一下那些开源 BIOS 的配置和编译过程，其使用方法暂时不做介绍。

8.1 编译 vboot

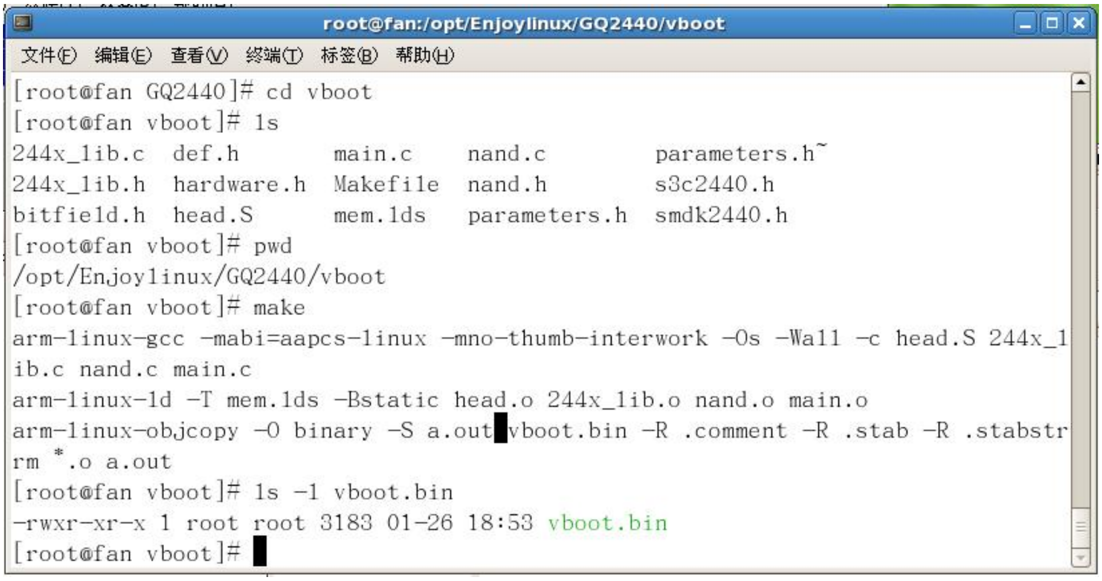
在第五章节中我们已经解压安装好了 vboot 的源代码，它位于/opt/Enjoylinux/GQ2440/vboot 目录中。

编译 vboot 十分简单，进入该目录，运行以下命令:

#cd /opt/Enjoylinux/GQ2440/vboot

#make

将会在当前目录下生成 vboot.bin，它和光盘中提供的是完全一样的目标文件，如图:



此时已经在当前目录下生成了 vboot.bin，您可以参考前面的章节把 vboot.bin 烧写到目标板的 Nand Flash 用以启动 Linux 系统。

8.2 配置和编译 vivi

注意：本开发板的所有 Linux 软件源代码均使用统一的编译器 arm-linux-gcc-4.3.2，其安装和设置见第五章节；本小节的 vivi 仅适合于 64M Nand Flash 版本的 GQ2440。

在第五章节中我们已经解压安装好了 vivi 的源代码，它位于**/opt/Enjoylinux/**

GQ2440/bootloader/vivi 目录中。

进入该目录，运行以下命令:

```
#cd /opt/Enjoylinux/ GQ2440/bootloader/vivi
#cp smdk2440 .config
#make menuconfig
#make
```

此时已经在当前目录下生成了 vivi，您可以参考前面的章节把 vivi 烧写到目标板的 Nand Flash 运行。

说明:

armrvivi，其实是由两个个程序组成的:

1. vboot 和 nboot 的混合体,
2. 改进的 vivi

他们使用不同的编译器编译出，最后粘在一起形成 armvivi，其中，vboot 和 nboot 的混合体暂且可以成为 xboot，它的原始版本是 nboot，但做了比较大的修改，以便可以做到这三个功能:

(1)检测开发板的 K1-K6 按键是否按下以启动 supervivi

(2)启动 Linux

(3)启动 WinCE

xboot 是由 ARM Linux GCC 编译的，其 bin 文件小于 4K，因为 xboot 的功能比较多，而大小有限，所以无法像 nboot 那样加载开机图片和进度条，像 vboot 或者 nboot 一样，xboot 实际是支持各个容量 nand flash 读取启动的。

原来三星的 vivi 如果要同时支持页大小 512 byte 和 2K byte 的 nand flash，需要做很多的调整，经我们改进的 vivi 可以同时支持 64M 和 128M，它具有一下特点:

(1)下载烧写 Linux 映像文件

(2)设置 Linux 启动参数

(3)格式化或者烧写整片 Nand Flash

(4)下载裸机程序到内存运行，或者烧写到 nand flash 的 block 0

至此，或许你应该明白，armvivi 实际是一团 bootloader 糅合在一起的东西，它无法通

过单一的编译器制作生成，对于它的开放性也就几乎失去了意义，armvivi 的功能很强的，而且简单易用，你只要学会使用它就可以了。

8.3 配置和编译译 U-Boot

说明：本小节只介绍 U-Boot 的配置编译和烧写，关于其使用方法的详细介绍，我们将会在以后的手册更新中添加，用户也可以自行到网上查找相关资料。

本光盘中的 U-Boot 具有以下功能特性:

1. 同时支持 S3C2410 和 S3C2440
2. 支持串口 xmodem 协议
3. 支持 USB 下载，可以在 PC 上使用 dnw 传数据
4. 支持网卡芯片 CS8900
5. 支持 NAND Flash 读写
6. 支持从 Nor/Nand Flash 启动
7. 支持烧写 yaffs 文件系统映象
8. 支持 lcd 驱动，在 uboot 运行时显示 logo
9. 既可以支持 CS8900，又可以支持 DM9000，但是，不能同时支持；要选择支持哪个网卡芯片，需要在 include/configs/GQ2440.h 中进行配置。

下面是具体的编译方法和烧写步骤。

8.3.1 配置和编译 U-Boot

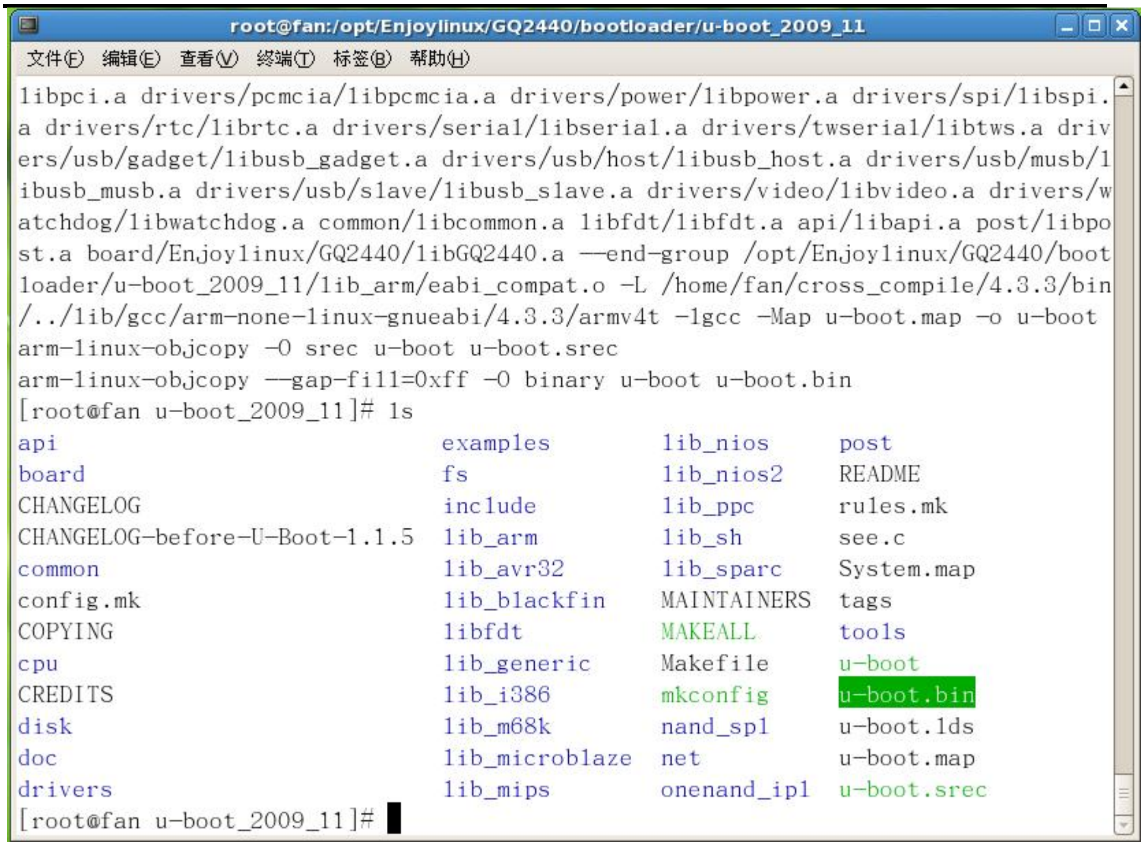
在第五章节中我们已经解压安装好了u-boot的源代码， 它位于**/opt/Enjoylinux/ GQ2440/bootloader/u-boot_2009_11** 目录中。

进入该目录，运行以下命令:

#cd /opt/Enjoylinux/ GQ2440/bootloader/u-boot_2009_11
#make distclean *;可以先做一次彻底的清理*
#make GQ2440_config *;配置U-Boot*
#make

就可以开始编译了，编译完毕，如图所示生成u-boot.bin

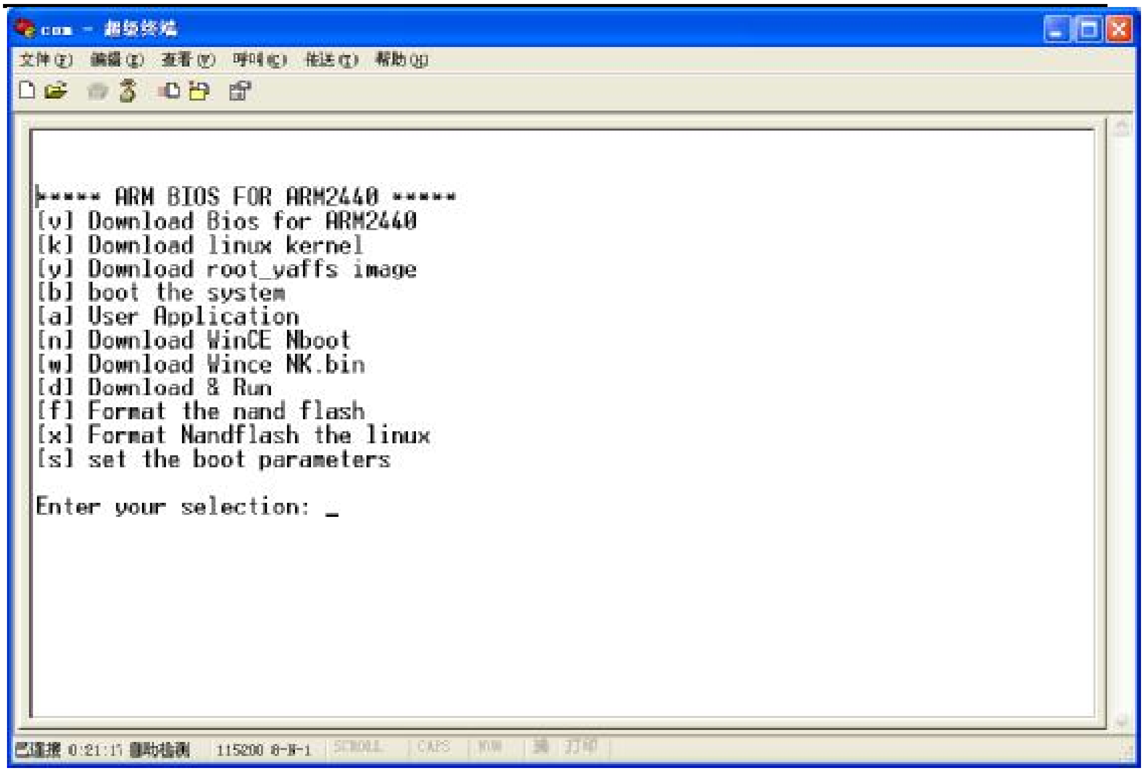
```
root@fan:/opt/Enjoylinux/GQ2440/bootloader/u-boot_2009_11
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@fan u-boot_2009_11]# pwd
/opt/Enjoylinux/GQ2440/bootloader/u-boot_2009_11
[root@fan u-boot_2009_11]# ls
api                            examples        lib_nios       post
board                          fs              lib_nios2      README
CHANGELOG                      include         lib_ppc        rules.mk
CHANGELOG-before-U-Boot-1.1.5  lib_arm         lib_sh         see.c
common                         lib_avr32       lib_sparc      System.map
config.mk                      lib_blackfin    MAINTAINERS    tags
COPYING                        libfdt          MAKEALL        tools
cpu                            lib_generic     Makefile       u-boot
CREDITS                        lib_i386        mkconfig       u-boot.bin
disk                           lib_m68k        nand_spl       u-boot.lds
doc                            lib_microblaze  net            u-boot.map
drivers                        lib_mips        onenand_ipl    u-boot.srec
[root@fan u-boot_2009_11]# make distclean
[root@fan u-boot_2009_11]# make GQ2440_config
Configuring for GQ2440 board...
[root@fan u-boot_2009_11]# make
```



```
root@fan:/opt/Enjoylinux/GQ2440/bootloader/u-boot_2009_11
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
libpci.a drivers/pcmcia/libpcmcia.a drivers/power/libpower.a drivers/spi/libspi.
a drivers/rtc/librtc.a drivers/serial/libserial.a drivers/twserial/libtws.a driv
ers/usb/gadget/libusb_gadget.a drivers/usb/host/libusb_host.a drivers/usb/musb/l
ibusb_musb.a drivers/usb/slave/libusb_slave.a drivers/video/libvideo.a drivers/w
atchdog/libwatchdog.a common/libcommon.a libfdt/libfdt.a api/libapi.a post/libpo
st.a board/Enjoylinux/GQ2440/libGQ2440.a --end-group /opt/Enjoylinux/GQ2440/boot
loader/u-boot_2009_11/lib_arm/eabi_compat.o -L /home/fan/cross_compile/4.3.3/bin
/../lib/gcc/arm-none-linux-gnueabi/4.3.3/armv4t -lgcc -Map u-boot.map -o u-boot
arm-linux-objcopy -O srec u-boot u-boot.srec
arm-linux-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
[root@fan u-boot_2009_11]# ls
api                         examples        lib_nios      post
board                       fs              lib_nios2     README
CHANGELOG                   include         lib_ppc       rules.mk
CHANGELOG-before-U-Boot-1.1.5  lib_arm      lib_sh        see.c
common                      lib_avr32       lib_sparc     System.map
config.mk                   lib_blackfin    MAINTAINERS   tags
COPYING                     libfdt          MAKEALL       tools
cpu                         lib_generic     Makefile      u-boot
CREDITS                     lib_i386        mkconfig      u-boot.bin
disk                        lib_m68k        nand_spl      u-boot.lds
doc                         lib_microblaze  net           u-boot.map
drivers                     lib_mips        onenand_ipl   u-boot.srec
[root@fan u-boot_2009_11]#
```

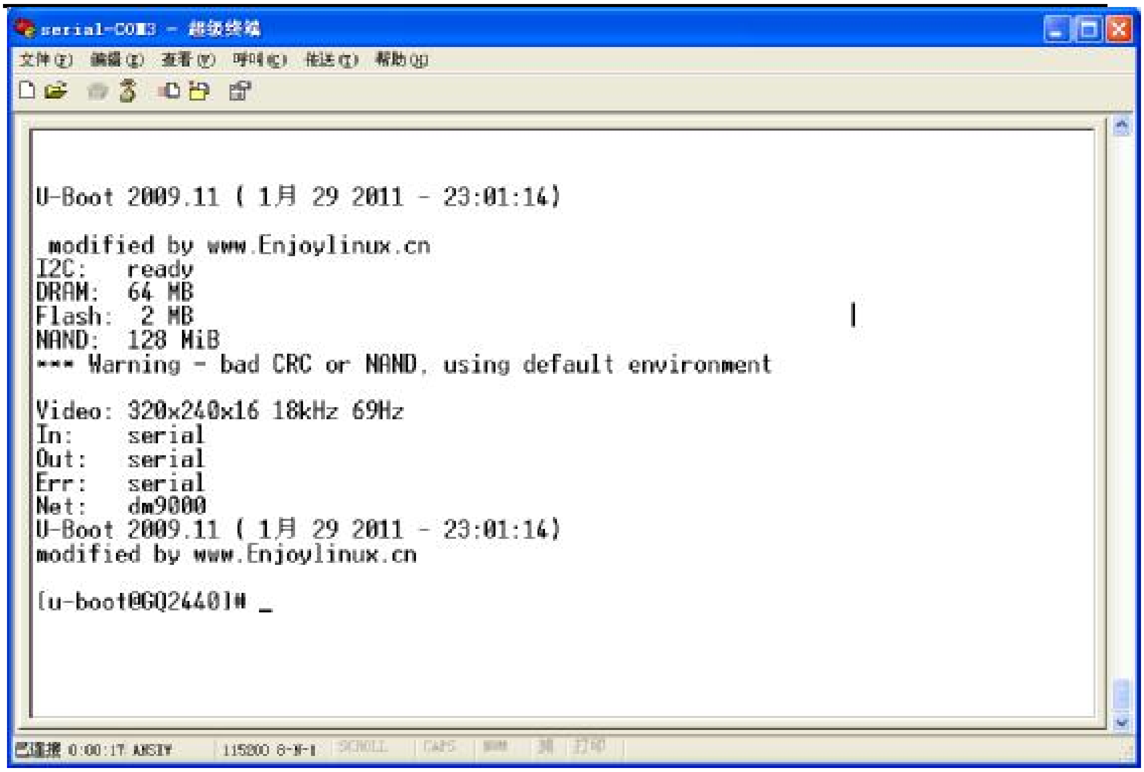
8.3.2 把 U-Boot 烧写到开发板

要烧写 U-Boot，需要通过Nor_KEY按钮使开发板从Nor Flash 启动，连接好串口和USB线，打开超级终端，打开电源，串口显示如图：



选择功能号“a”，打开DNW，确认USB 连接正常OK，点【UsbPort】|【Transmit/Restore】，选择刚才所编译的**u-boot.bin**，下载和烧写很快就会结束。

通过POW_KEY按钮使开发板从Nand Flash 启动，重新复位或者重启开机电源开关，在串口终端可以看到如图信息:



在运行uboot时lcd显示出我们的logo:



由于时间原因，目前uboot只支持命令行模式，不支持菜单模式，下面将介绍如何通过命令行启动linux。

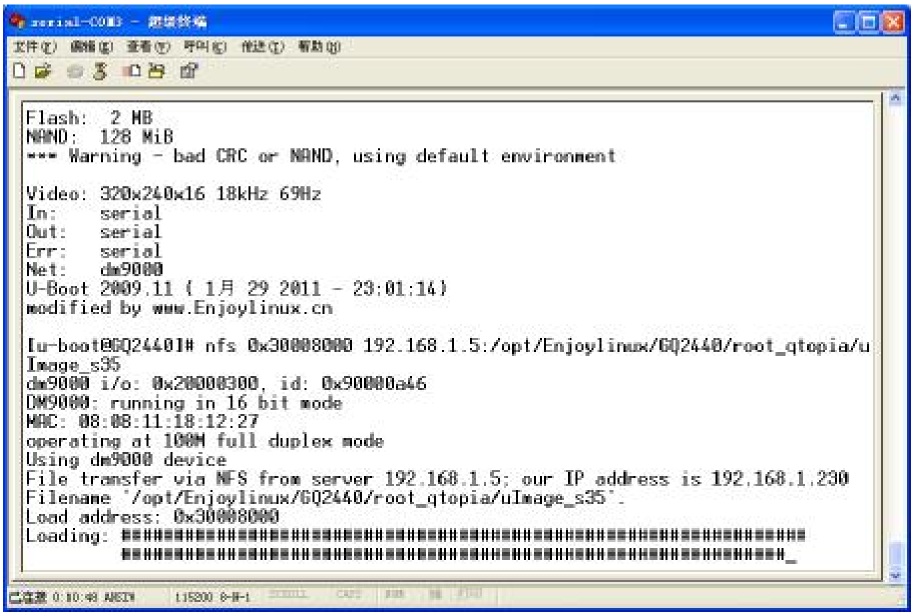
8.3.3 通过 U-Boot 启动 linux 系统

在命令行中输入命令：

#nfs 0x30008000 192.168.1.5:/opt/Enjoylinux/GQ2440/root_qtopia/uImage_s35

可以通过nfs将**/opt/Enjoylinux/GQ2440/root_qtopia/**中的linux镜像**uImage_s35**下载到内存地址空间**0x30008000**处。其中，**192.168.1.5**是nfs服务器（也就是pc机）的ip地址。

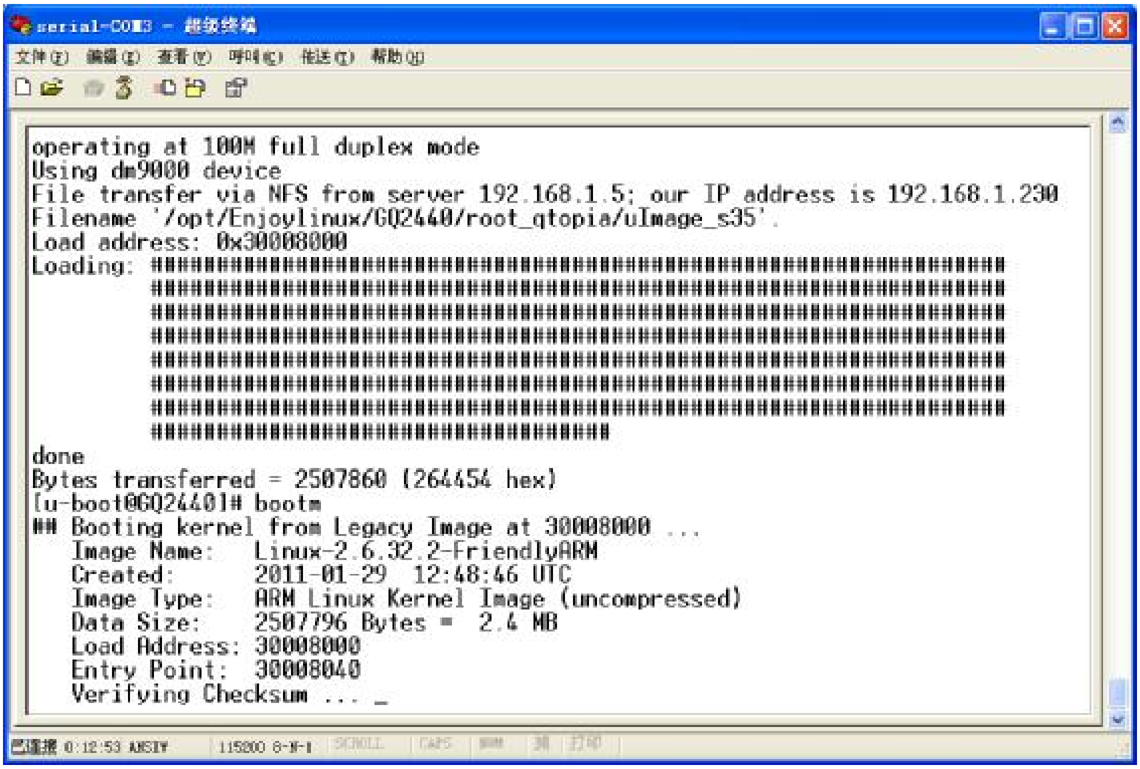
说明： uImage_s35镜像在光盘目录**images/limux/**目录下，需要事先拷贝到**/opt/Enjoylinux/GQ2440/root_qtopia/**目录中。



下载完毕后，通过命令：

#bootm

启动 linux 系统：



注意：uImage 是 uboot 启动的 linux 镜像，需要通过在 linux 内核根目录下使用编译命令：#make uImage 得到。

8.3.4 设置 U-Boot 自动启动 linux 系统

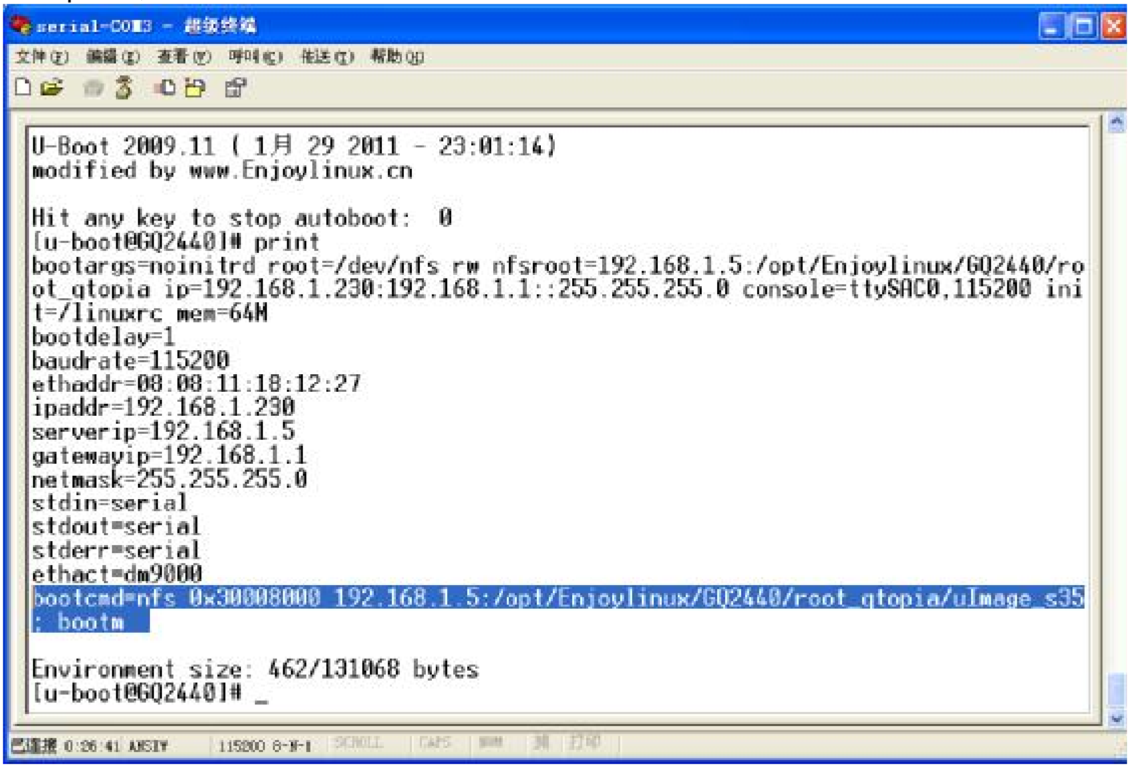
在命令行输入命令：

#setenv bootcmd "nfs 0x30008000 192.168.1.5:/opt/Enjoylinux/GQ2440/root_qtopia/uImage_s35; bootm"

#save

设置启动变量 bootcmd，这样开机在一段延时后（如果在倒计时期间按下键盘任意键则 uboot 会进入命令行模式），uboot 就会自动执行 bootcmd 指定的命令下载并启动 linux 系统了。

使用 print 命令，能够查看变量 bootcmd 是否设置成功。



第9章 WindowsCE 5.0 开发指南

9.1 基于 WindowsCE5.0 的开发环境

本小节介绍如何建立 WindowsCE 5.0 的开发环境，这包括安装 Platform Builder 5.0 及其补丁，导入开发板所用的 BSP 和编译示例工程。

说明：Platform Builder 5 不能在 Windows Vista/2007 平台上安装使用，以下操作环境均基于 Windows XP。

9.1.1 安装 Platform Builder 5.0(含 2007 最新补丁)

注意：Windows Embedded CE 5.0 的安装文件，用户可以到微软网站自行下载它的试用版。

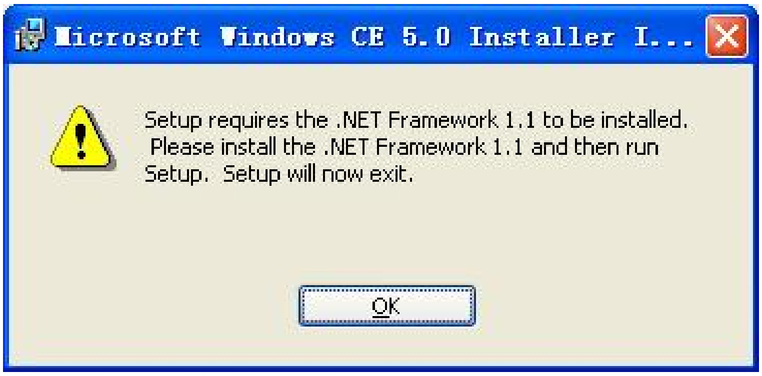
- **试用版下载地址：**
  http://www.microsoft.com/downloads/details.aspx?FamilyID=486E8250-D311-4F67-9FB3-23E8B8944F3E&displaylang=en
- **Platform Builder 5.0 的2008 补丁下载地址：**
  http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=6dccd9fc-a7ac-4fa3-a9bd-fcc7a83f5311

说明：以下安装步骤使用的补丁文件是 2007 年的，但此补丁已经在微软网站无法找到下载链接，微软建议直接使用2008 年的补丁，我们尚未验证2008 的补丁是否可用，Platform Builder 5.0 之2007 年的补丁安装文件位于光盘"\WindowsCE5.0\PB5 补丁 2007"目录。

安装提示：安装 PB5 时请把安装文件复制到硬盘安装，否则有可能会不能顺利执行以下步骤。

本节以图例介绍如何在 WindowsXP 上安装Platform Builder 5.0(简称PB5)，它用来开发和定制WINCE 内核，并可以用来调试内核等，安装PB 一般需要5-7G 的硬盘空间。

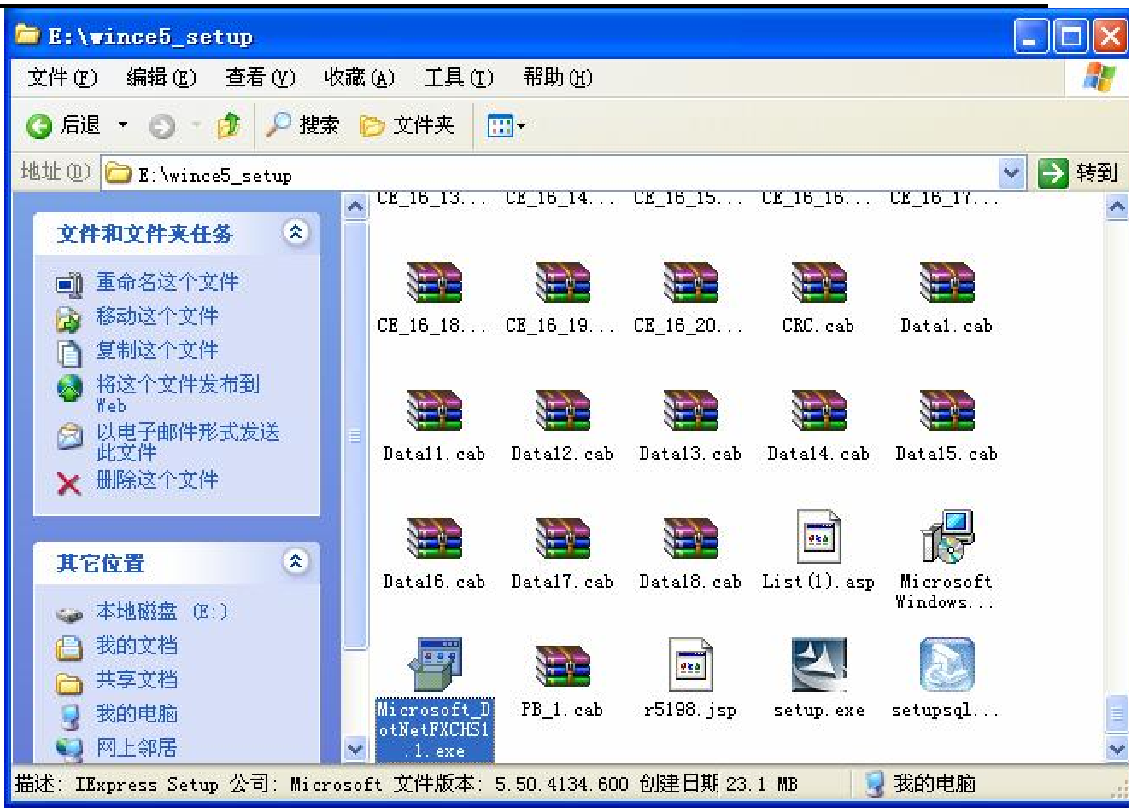
Step1: 安装PB5需要dotnet framework1.1，如果系统中没有安装此组件，将会出现如图提示：



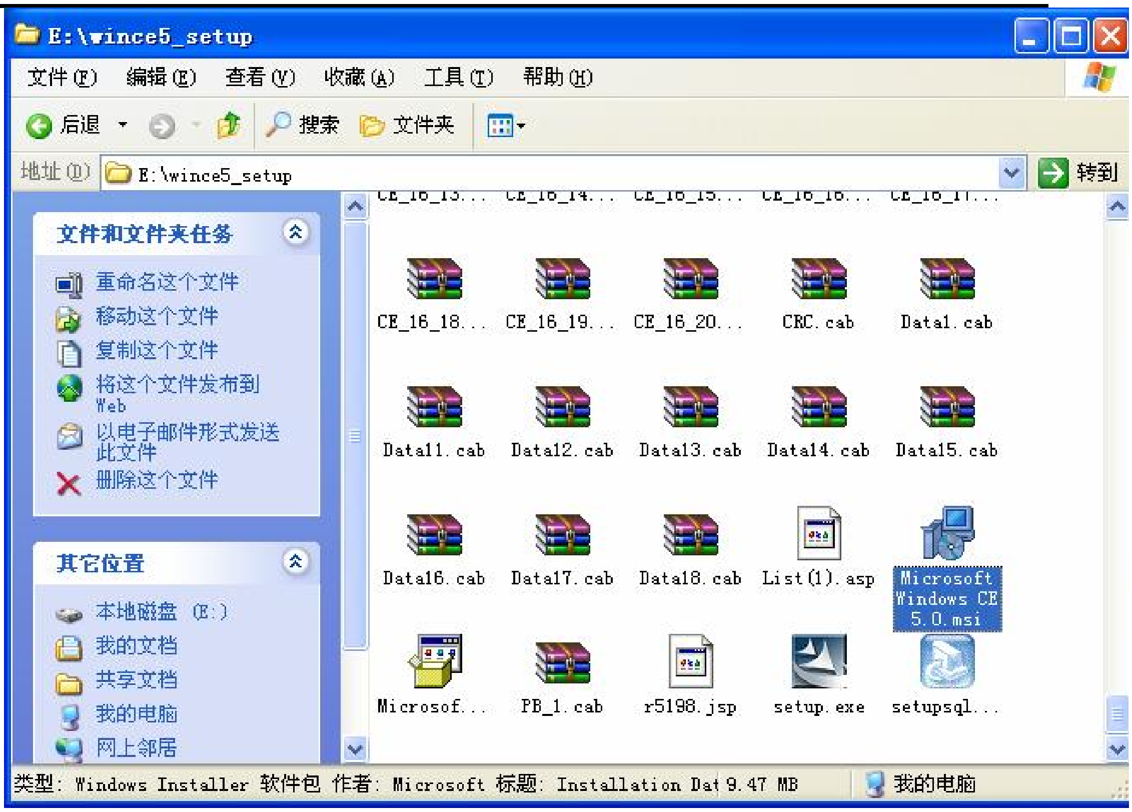
可以到如下网址下载.NET Framework 1.1

http://www.microsoft.com/downloads/details.aspx?displaylang=zh-cn&FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3

下载完毕后，双击运行按照提示安装即可，如图：



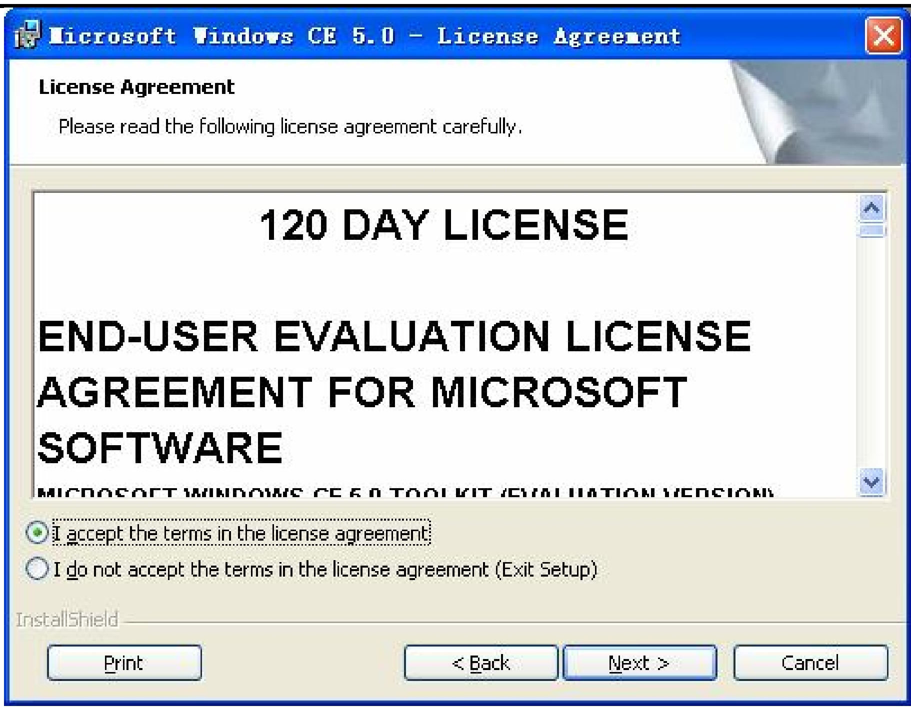
Step2: 在PB5安装光盘中找到Microsoft Windows CE 5.0.msi.exe，双击运行开始安装PB5:



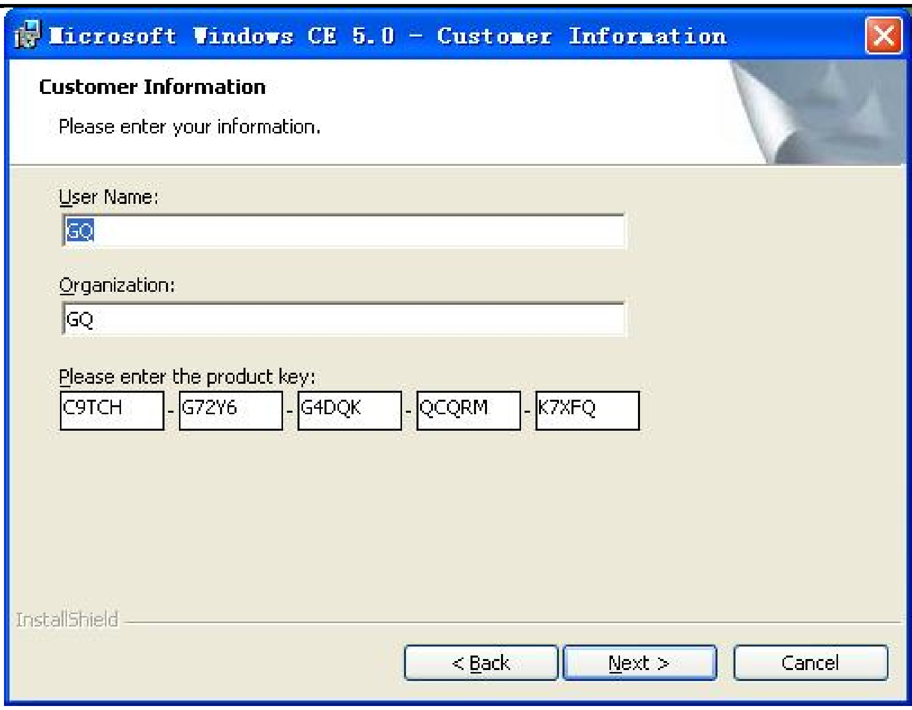
Step3: 出现安装向导窗口，点"Next"继续：



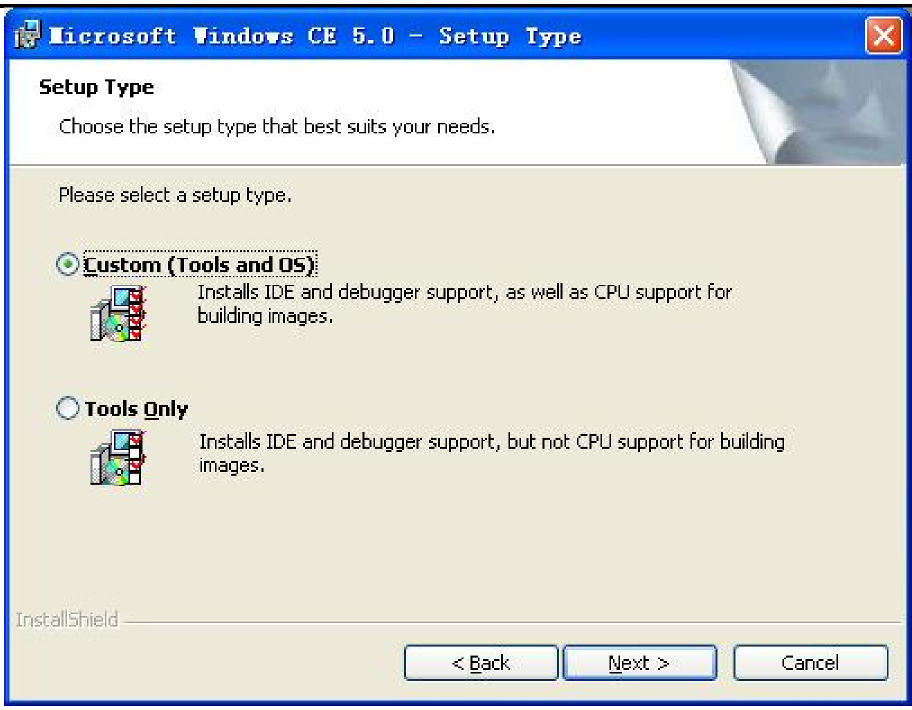
Step4: 出现“License Agreement”窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：



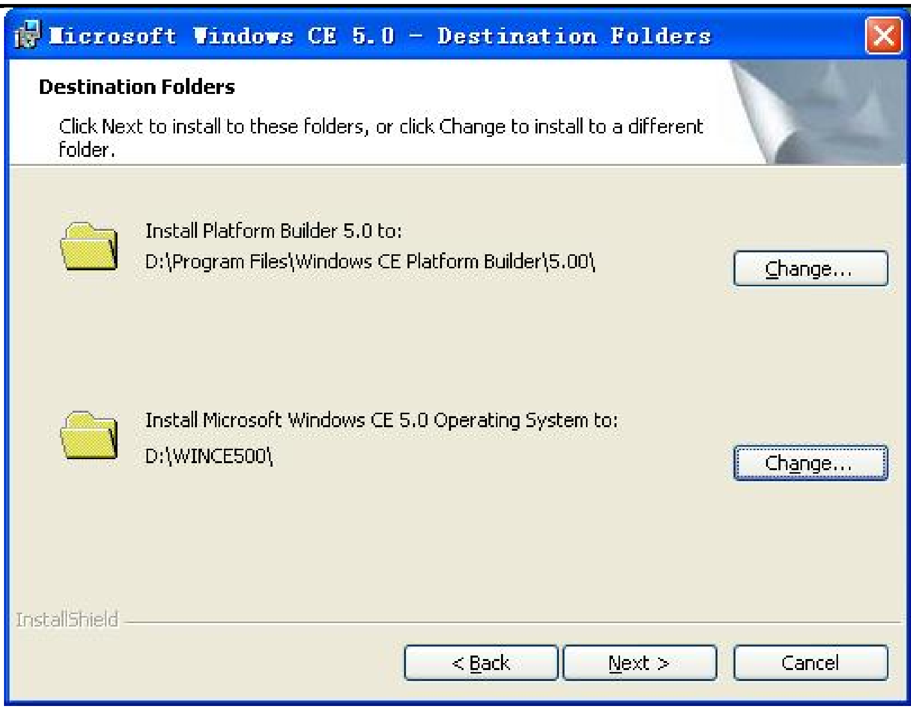
Step5: 输入用户信息和序列号，点“Next”继续：



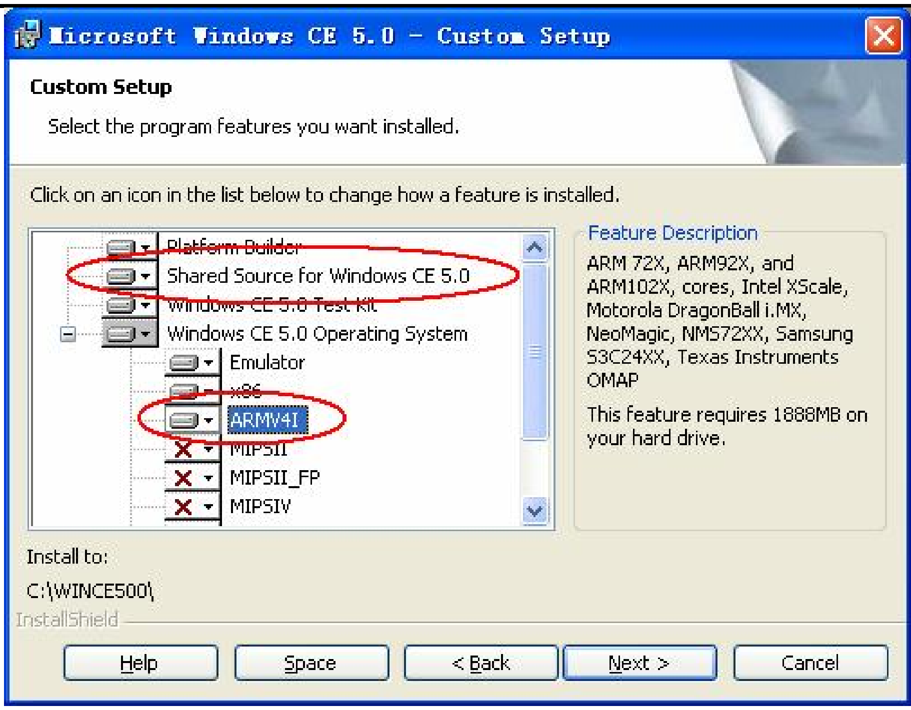
Step6: 选择安装类型为定制安装，如图，点“Next”继续



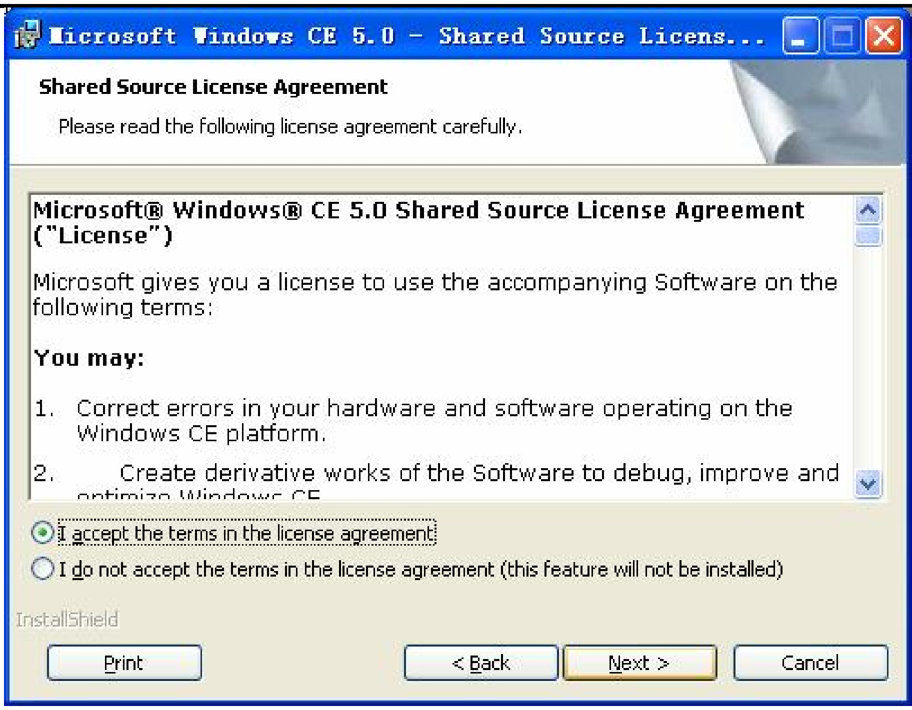
Step7: 选择安装目录，这里按修改为D盘，点“Next”继续



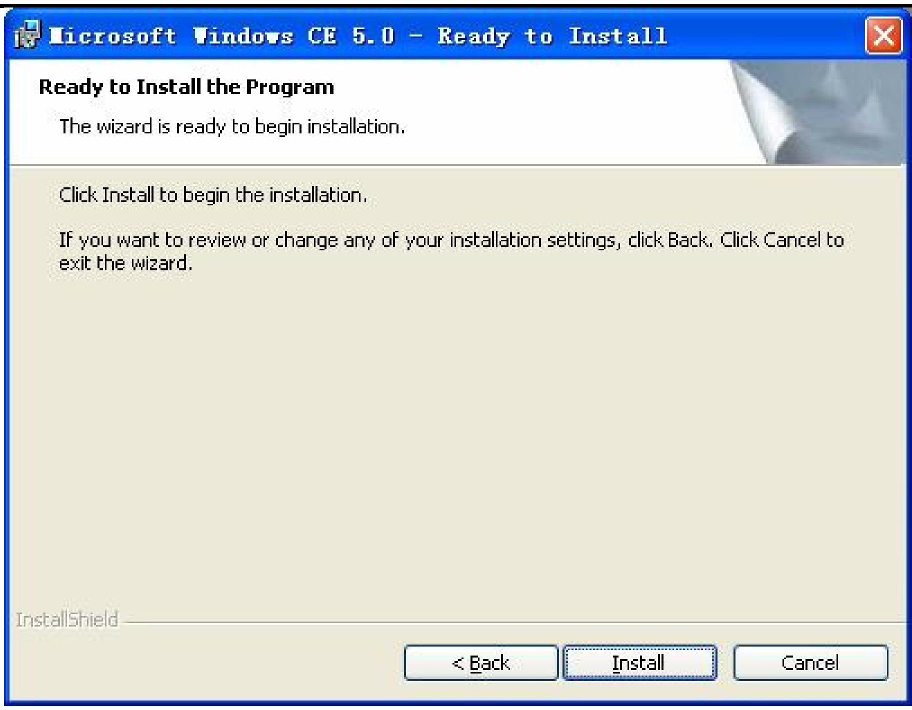
Step8: 在定制安装中选择您所需要的系统平台，在这里一定要选择安装ARMV4I，最好也选择安装“Shared Source for Windows CE 5.0”，如图.点"Next"继续



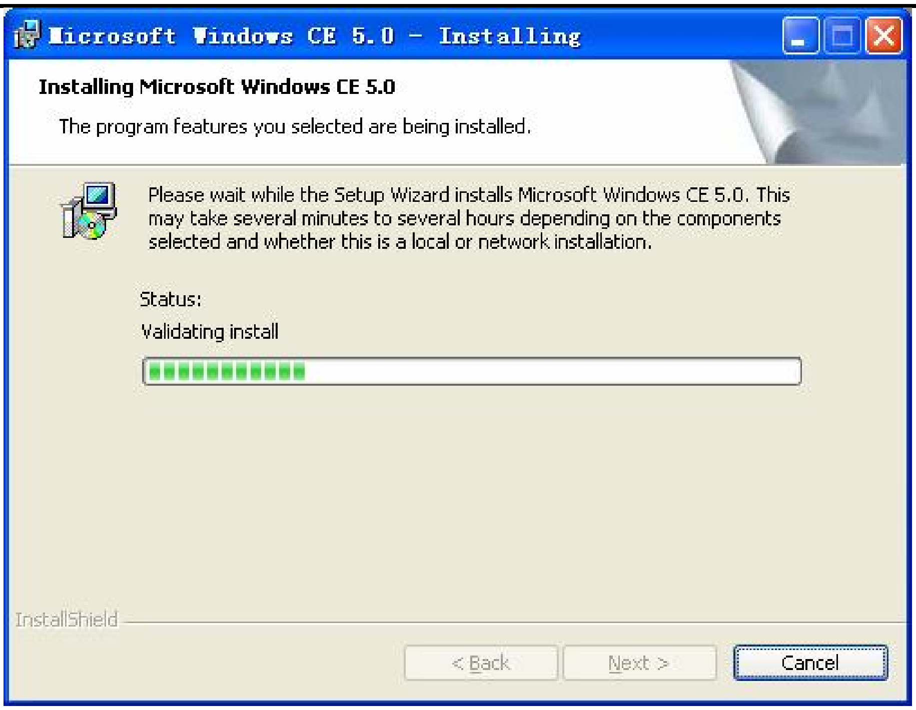
Step9: 出现许可协议窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：



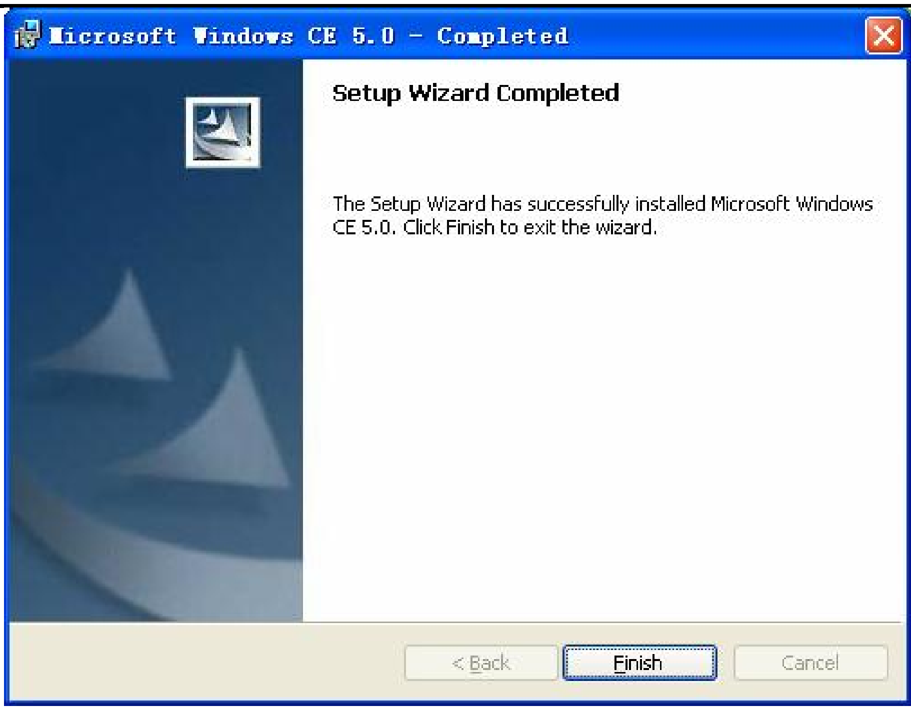
Step10: 出现如图提示窗口，点“Next”继续



Step11: 进入安装进程界面，此过程时间较长，请耐心等待，如图



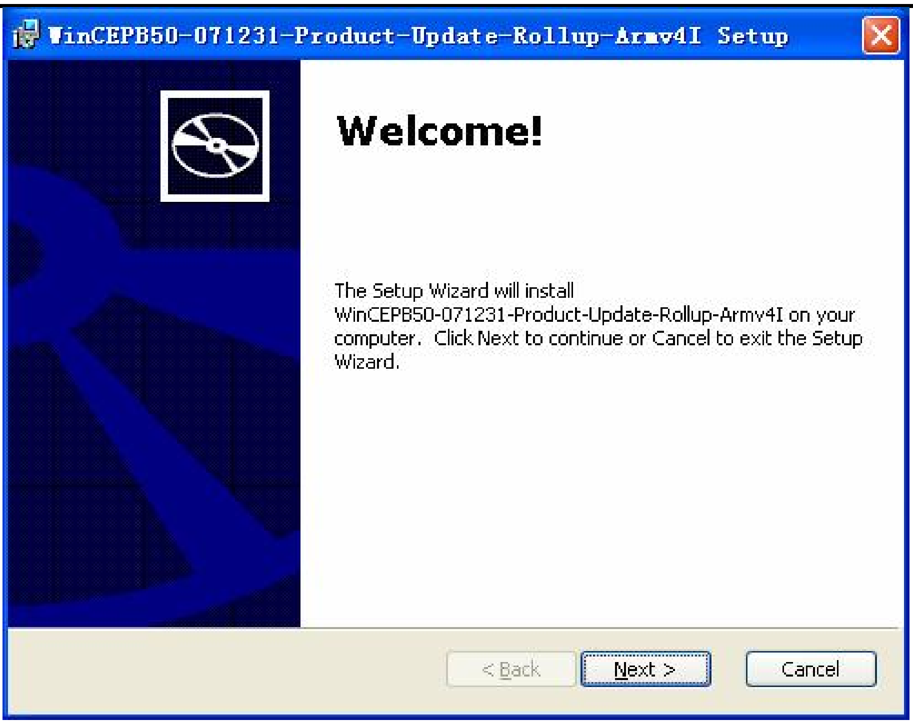
Step12: 安装完毕，如图



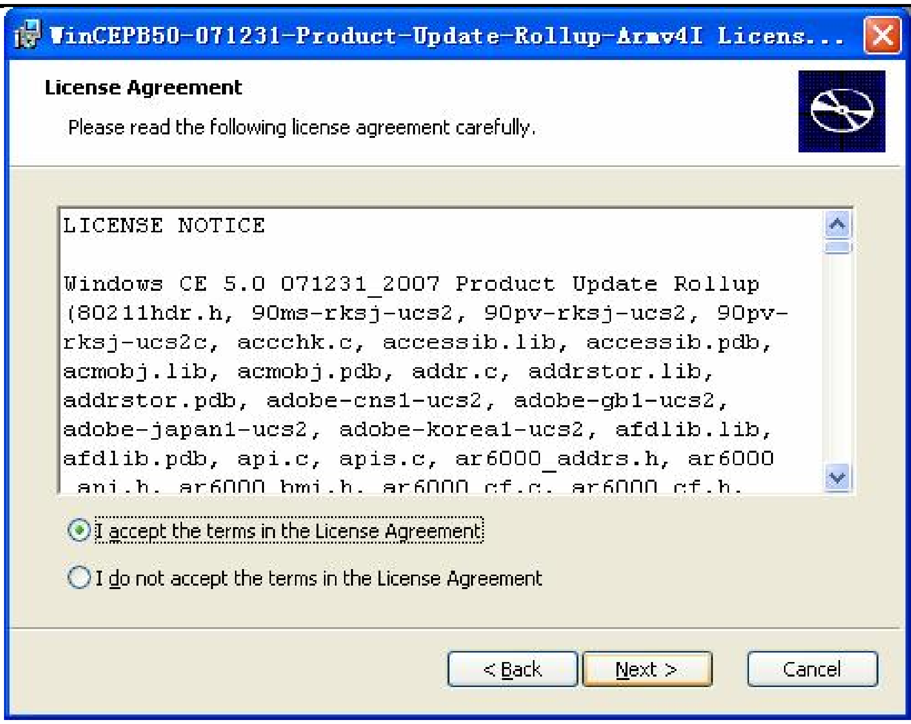
Step13: 现在开始安装PB5补丁程序，其安装文件位于光盘**WinCE源代码和软件\WindowsCE5.0\PB5补丁**文件夹中，双击该程序开始安装：



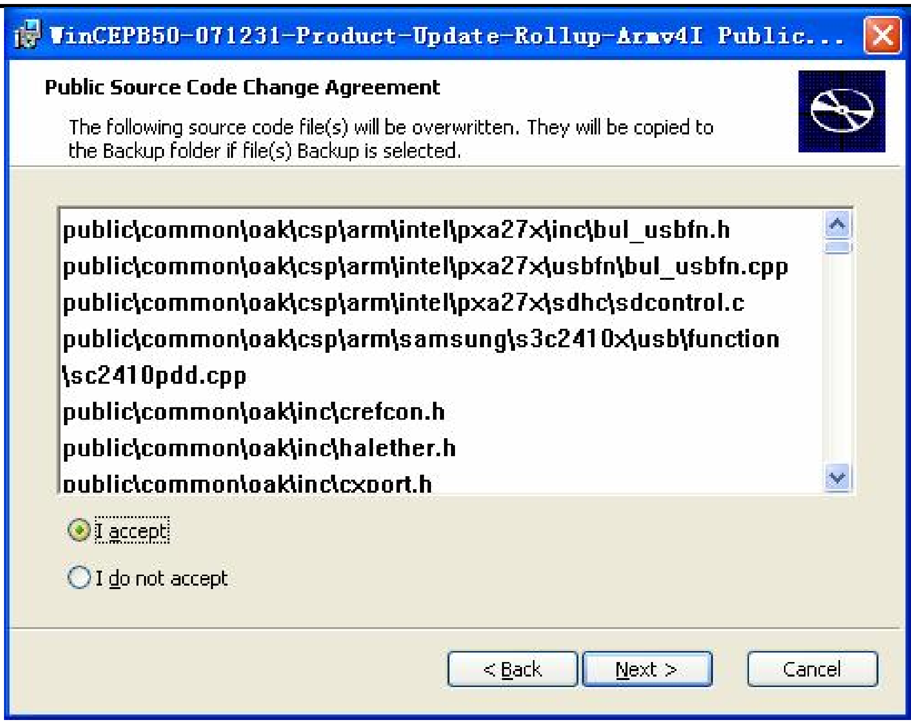
Step14: 出现安装向导界面，如图，点“Next”继续



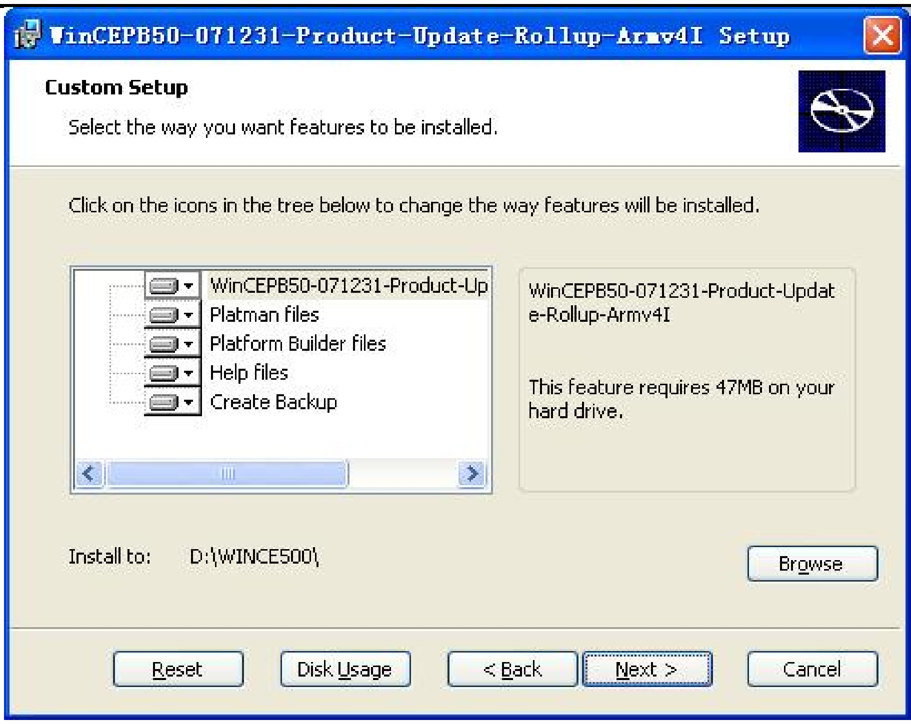
Step15: 出现许可协议窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：



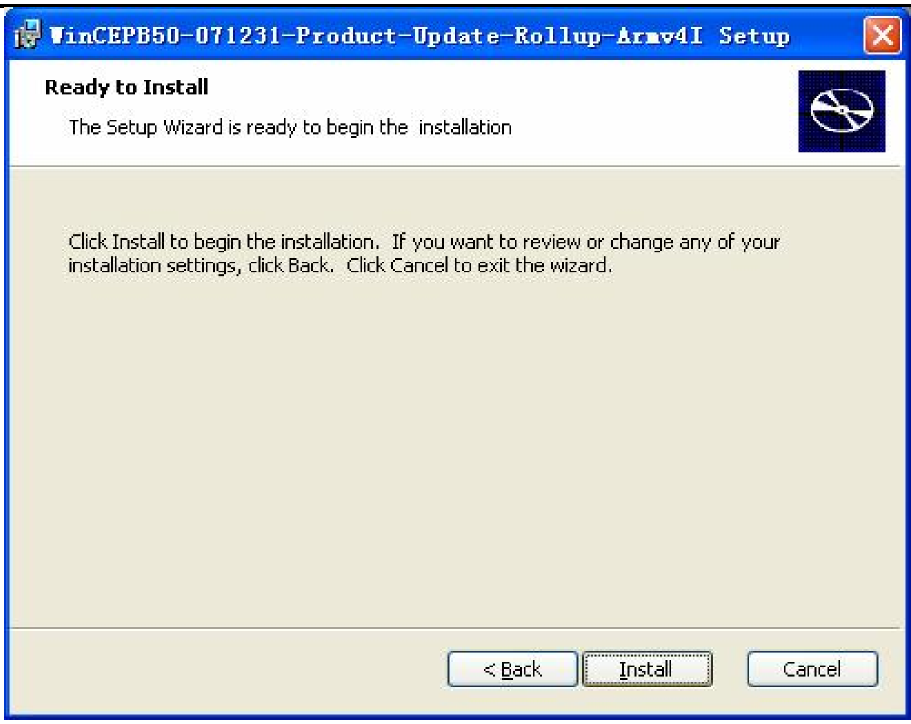
Step16: 出现如图公共代码许可协议窗口，选择“I accpet”，点“Next”继续



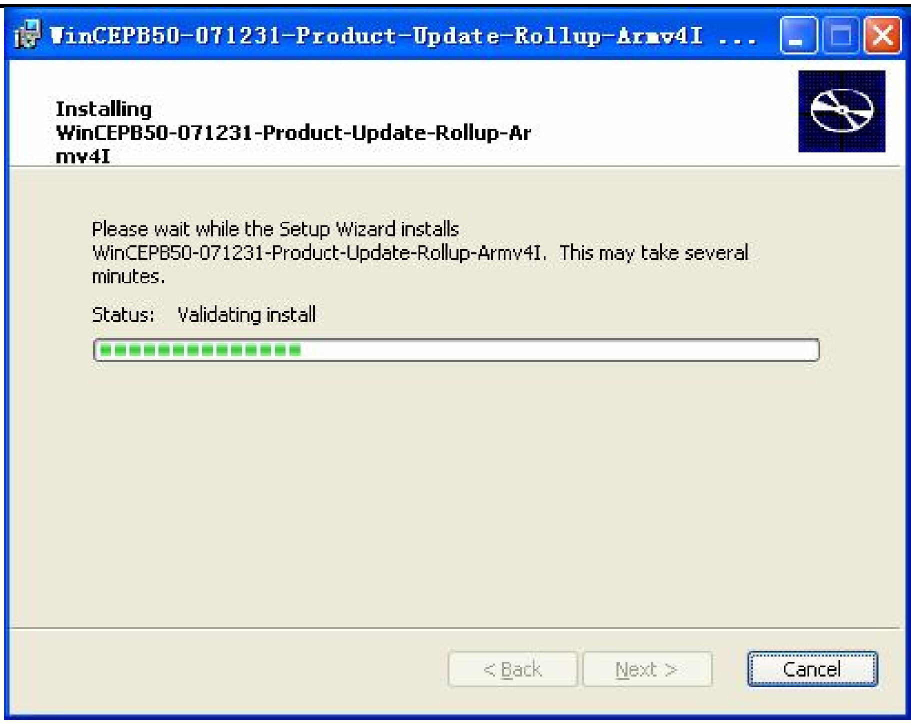
Step17: 出现定制安装界面，选择默认安装全部组件，点“Next”继续



Step18: 准备好安装，点“Next”继续



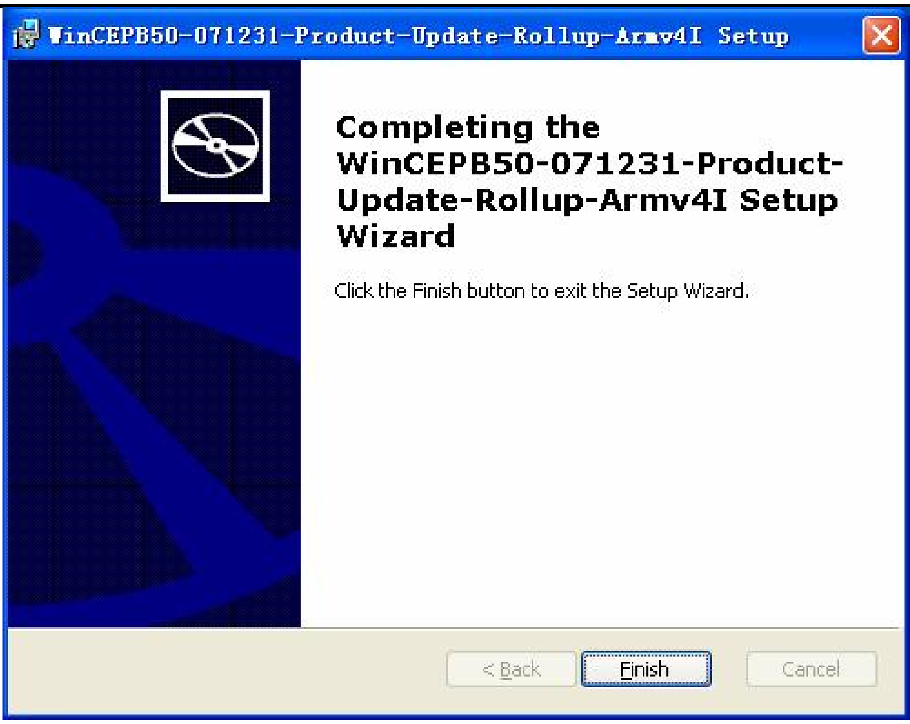
Step19: 安装过程如图所示，该过程比较长，请耐心等待



Step20: 安装过程会出现如下提示界面，不必理会，点"确定"即可



Step21: 安装完毕，如图



9.1.2 导入安装 BSP

WindowsCE5 的BSP 位于光盘目录：**WinCE源代码和软件\WindowsCE5.0\GQ2440 _10_06_BSP**

其中目录最后面的数字为发行或者更新日期，为了描述方法，我们以下把该目录简称为**GQ2440**。

说明： BSP 目前支持以下型号的液晶屏：

- 三星3.5寸屏带触摸，简称为s35
- 群创4.3寸屏带触摸，简称为a43
- 群创5.6寸屏带触摸，简称为a56
- 群创7寸屏带触摸，简称为a70
- VGA 模块显示输出，分辨率1024x768，简称为VGA1024x768

通过修改**GQ2440\Src\Inc\options.h** 头文件中LCD_TYPE 的定义，可以选择相应的

LCD 类型：

```
#define LCD_s35
//#define LCD_a43
//#define LCD_a56
//#define LCD_A70
//#define LCD_VGA1024768
```

光盘中缺省LCD 型号是LCD_s35。

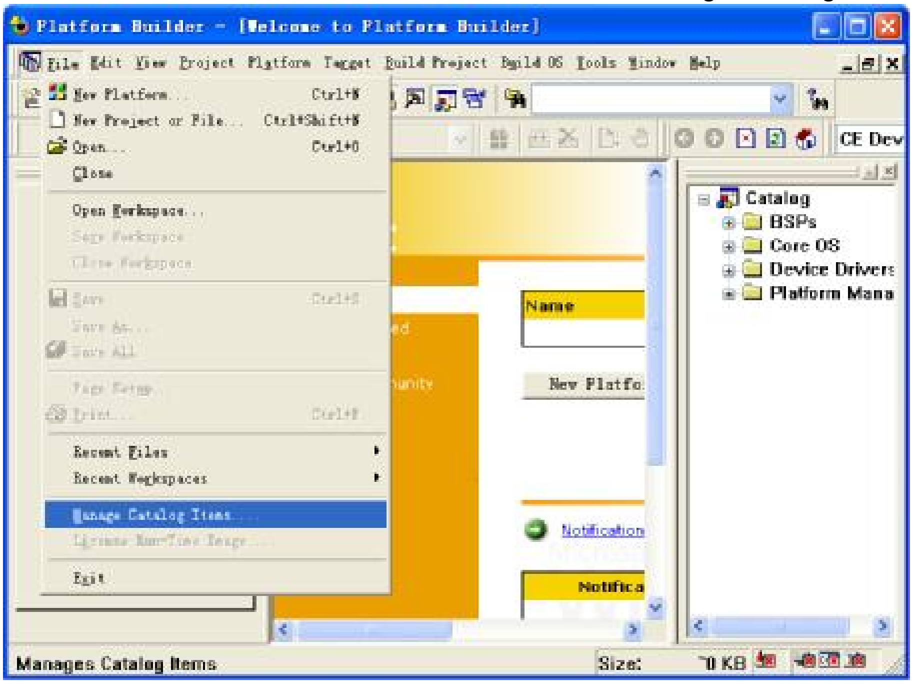
要使用 PB5 编译WINCE 内核映象，需要安装对应目标板的BSP，并进行一些设置，请按照以下步骤安装 BSP:

Step1: 把光盘**WindowsCE 5.0\GQ2440**文件夹复制到**C:\WINCE500\PLATFORM** 目录下，并去掉只读属性。

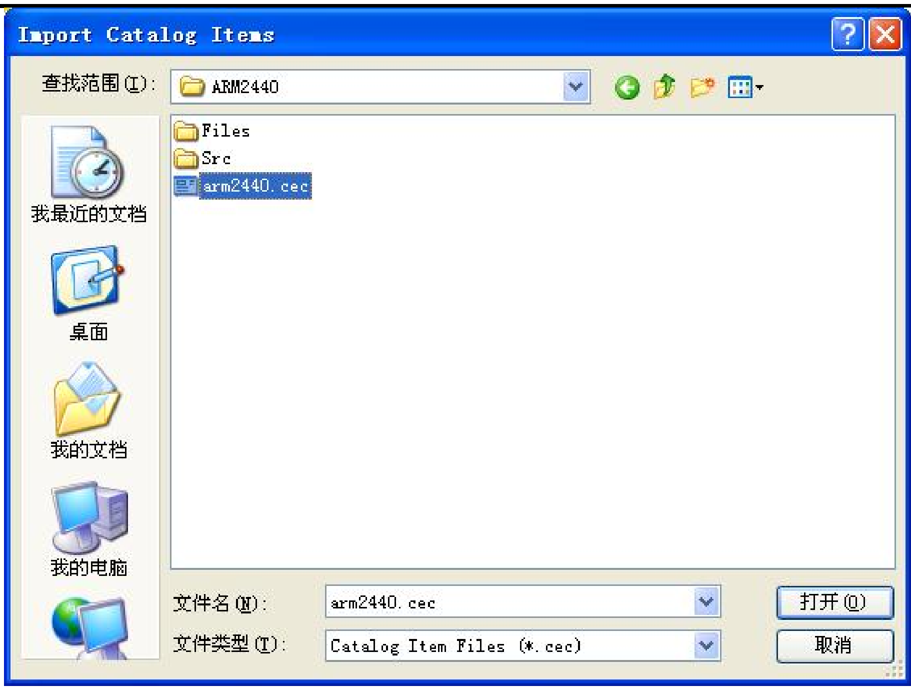


Step2: 通过【开始】|【所有程序】|【Microsoft Windows CE 5.0】|【Platform Builder

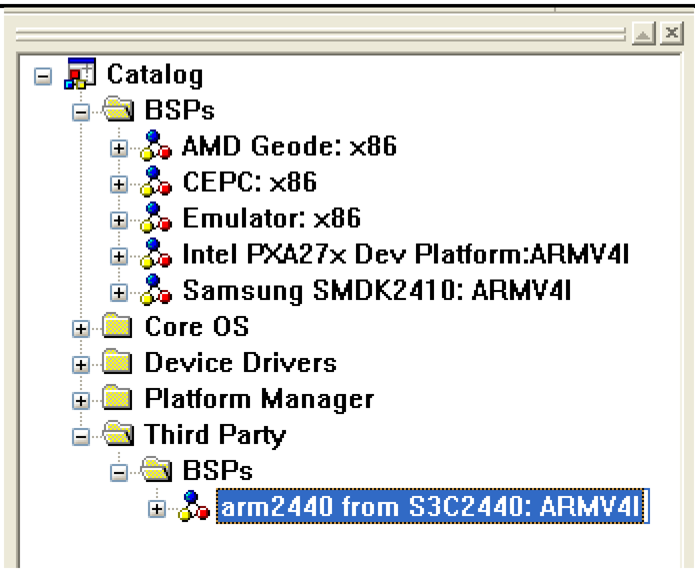
5.0】，打开“Platform Builder 5.0”，选择“File”菜单下的“Manage CatalogFeatures”



点“Import按钮”，导入“platform\GQ2440\GQ2440.cec”文件



Step3: 在“Catalog”的Third Party下，将会自动添加“GQ2440 from S3C2440: ARMV4I”项，BSP安装完成。



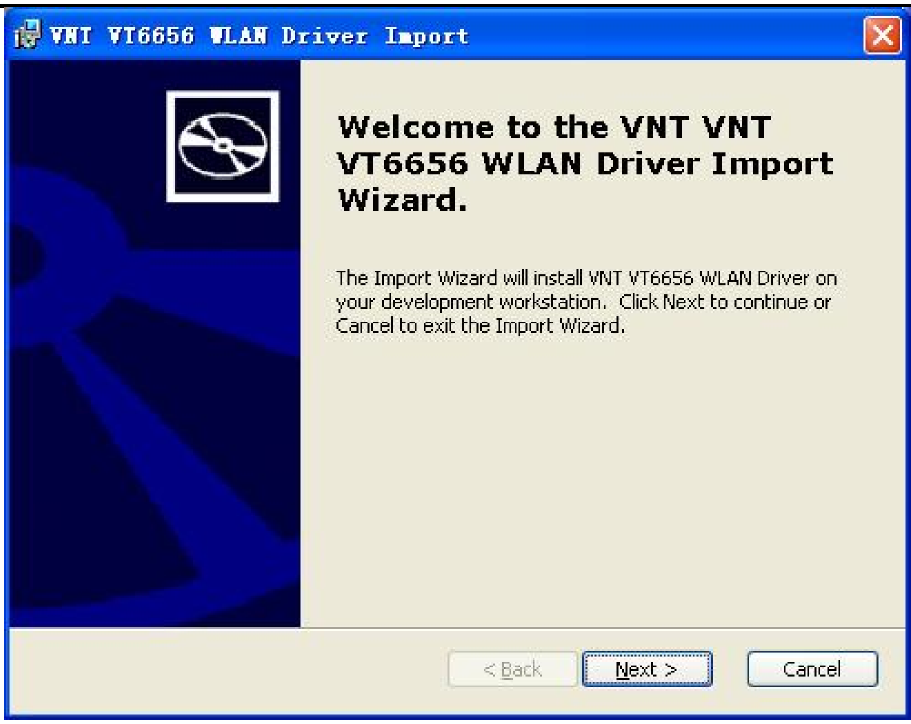
9.1.3 安装无线网卡驱动程序

说明: 无线网卡驱动并不是必须要安装的，如果不安装，使用 GQ2440.pbxml 工程示例编译出的 WinCE 内核将不具备无线网卡驱动。

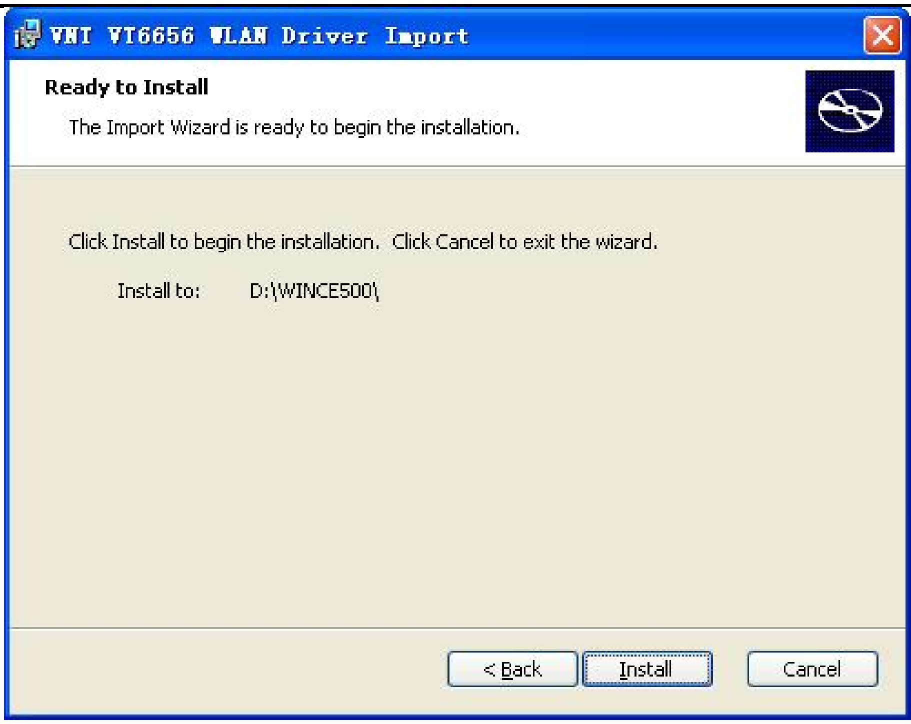
另外，此无线网卡驱动仅支持 VNT-6656G6A40 型号的 USB 无线网卡

无线网卡驱动程序位于光盘 **WinCE 源代码和软件\WindowsCE 驱动程序模块\无线网卡**文件夹中，它是一个安装文件 **VNUWLC5-ARM.msi**。下面是安装步骤:

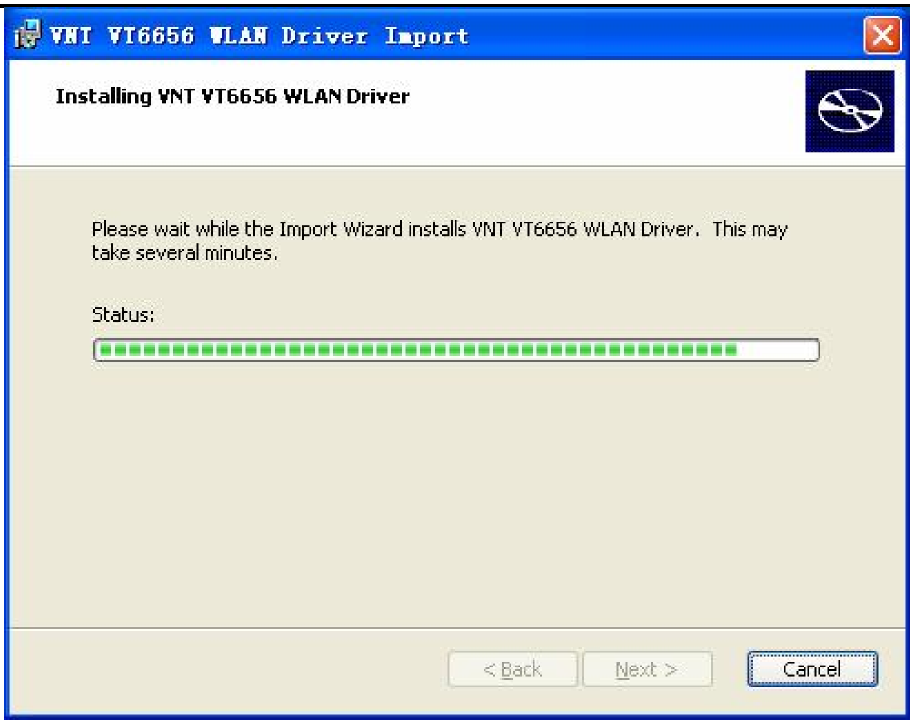
Step1: 双击运行安装程序，打开安装向导，点"Next"继续



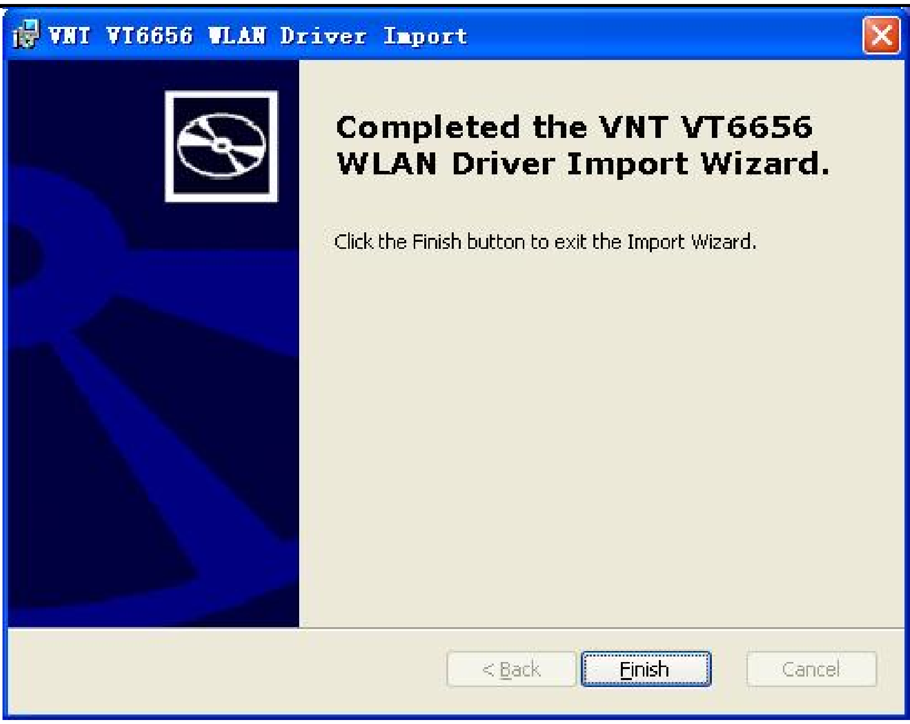
Step2: 开始安装，并提示将安装到D:\WINCE500目录中，如图



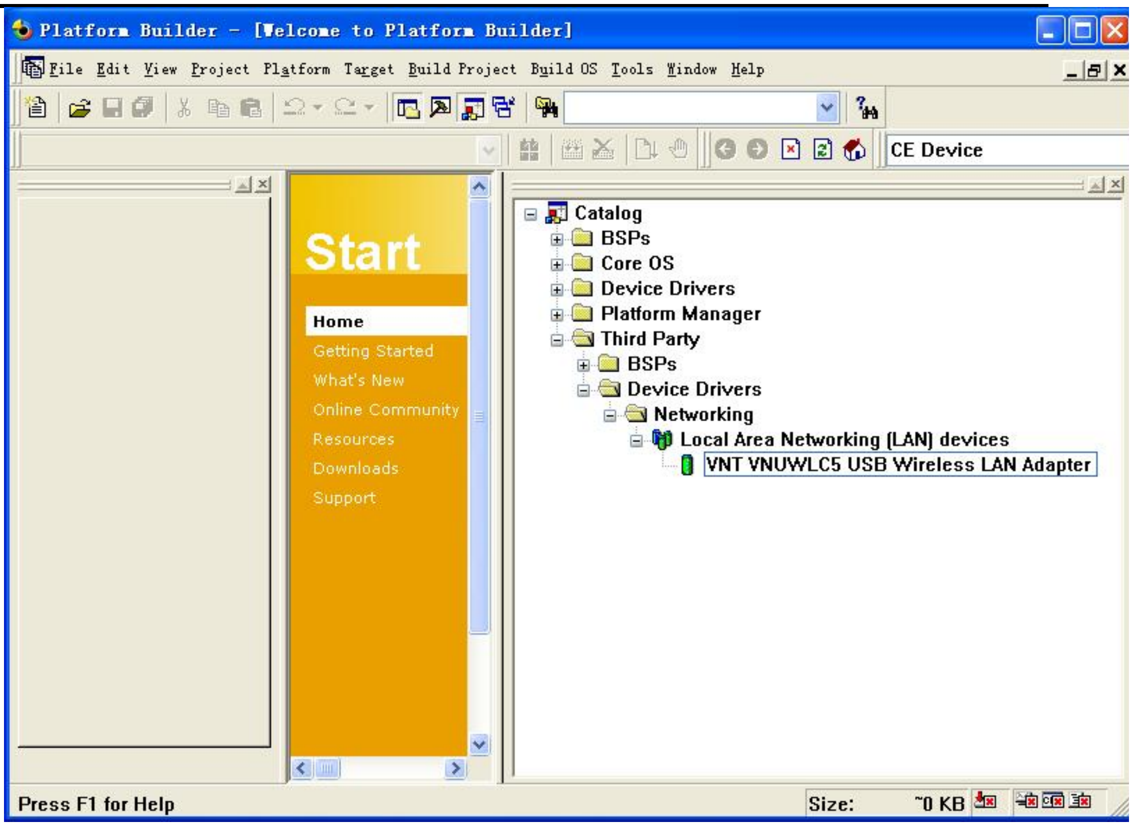
Step3: 安装进程如图，安装很快就会结束



Step4: 安装结束，点“Finish”结束安装

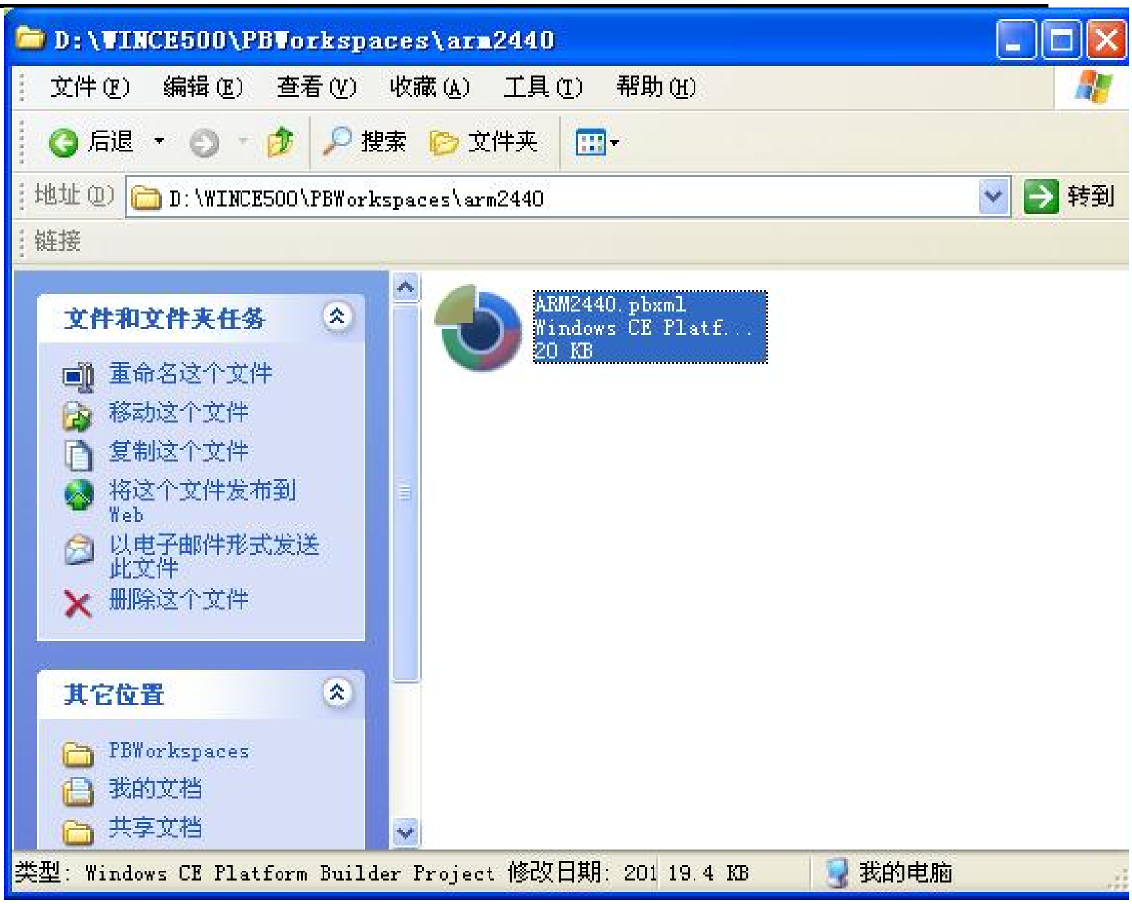


Step5: 这时打开PB5，会看到Catalog一栏出现如图选项，如图



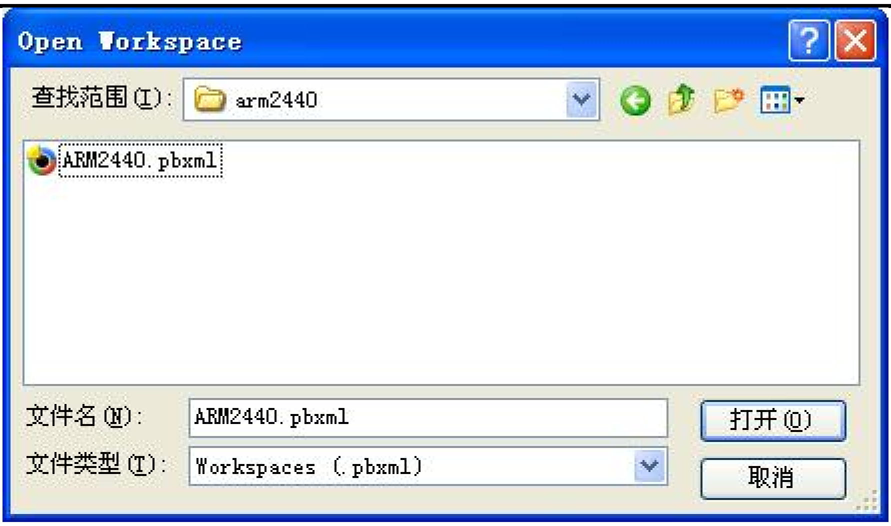
9.1.4 编译内核工程示例

Step1: 在C:\WINCE420\PBWorkspaces 目录(如果没有，可以手工创建一个)中创建一个文件夹“GQ2440”，把光盘中WindowsCE 5.0 目录下的GQ2440.pbxml 文件复制到D:\WINCE420\PBWorkspaces\GQ2440 目录，并去掉只读属性。

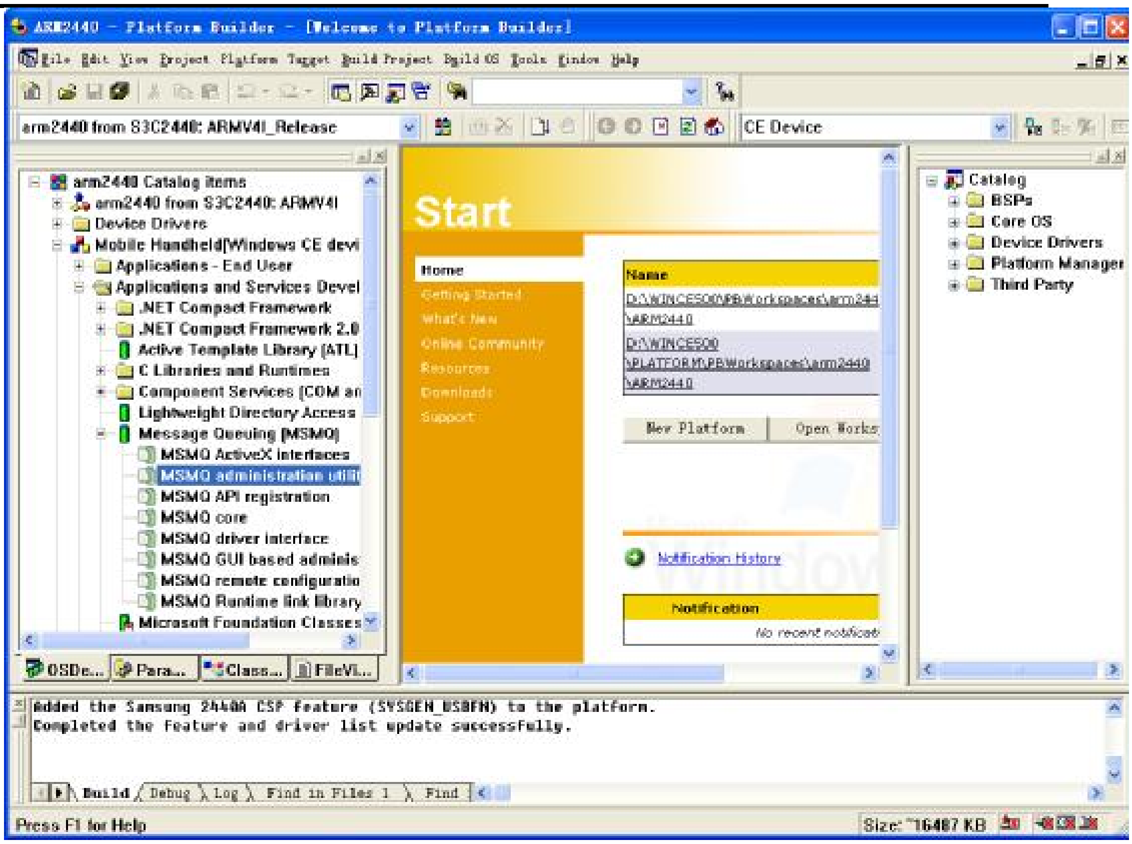


Step2: 在PB 中点【File】|【Open Workspace…】，打开刚刚复制的项目文件，注意是pbxml结尾的。

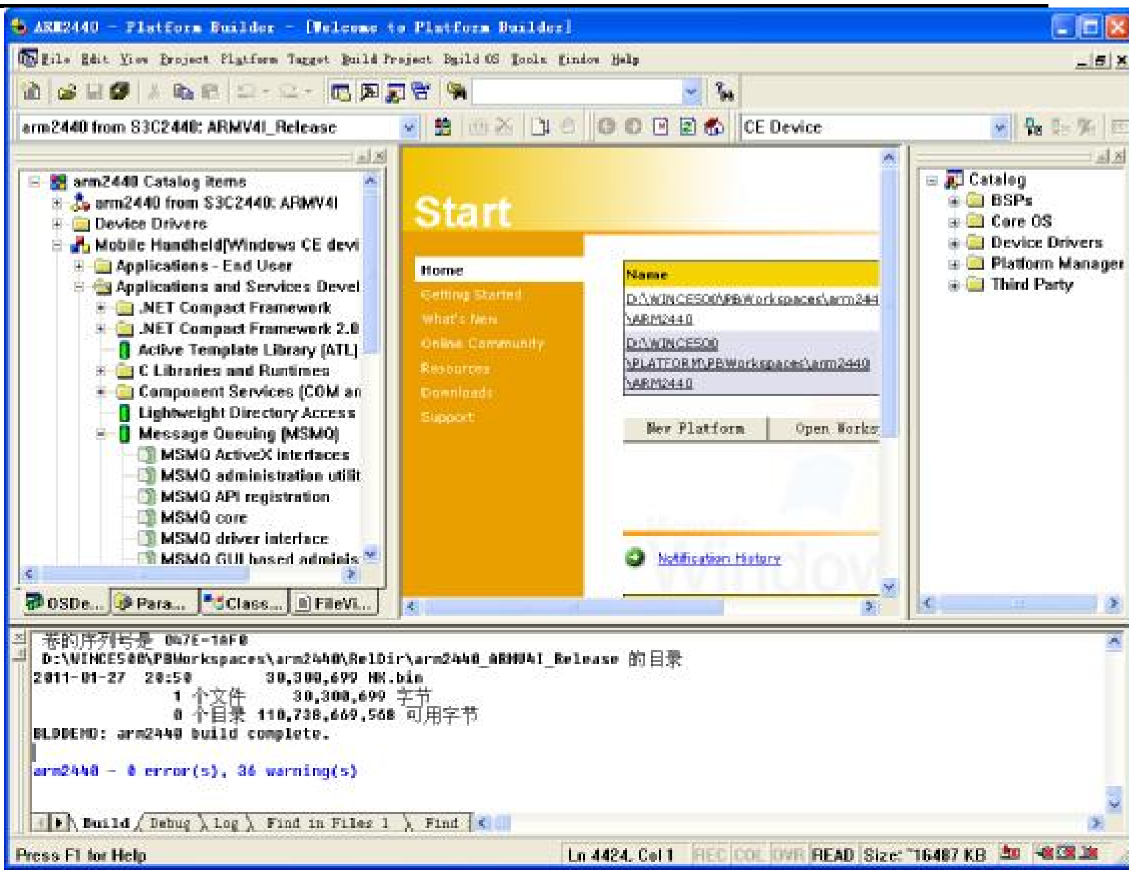
说明：也可以通过双击打开 GQ2440.pbxml 文件。



如图为打开的项目文件之 PB 界面:

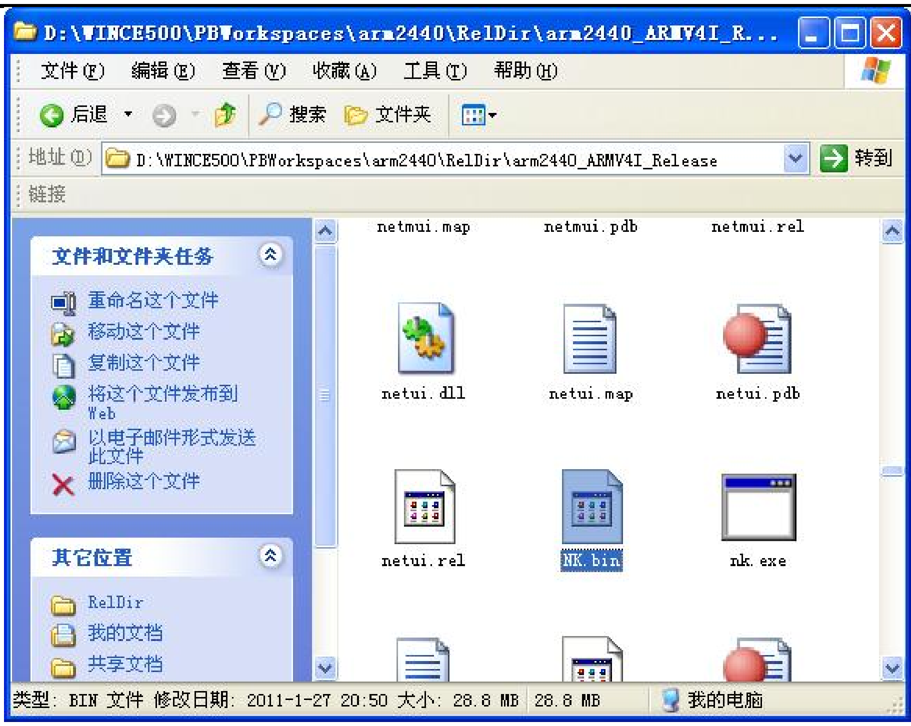


Step3: 打开后，点【Build OS】|【Sysgen】开始编译，或者点工具栏的图标开始进行编译，该过程比较长。



编译过程中有可能会出现如图这样的警告信息，这个是正常的，不必理会。

Step4: 编译完毕，就会生成“nk.bin”和“nk.nb0”两个文件，其中nk.bin 是发行版本，nk.nb0 是内存中运行版本， 我们一般使用nk.bin 。它们位于**D:\WINCE500\PBWorkspaces \GQ2440\RelDir\GQ2440_ARMV4I_Release.**



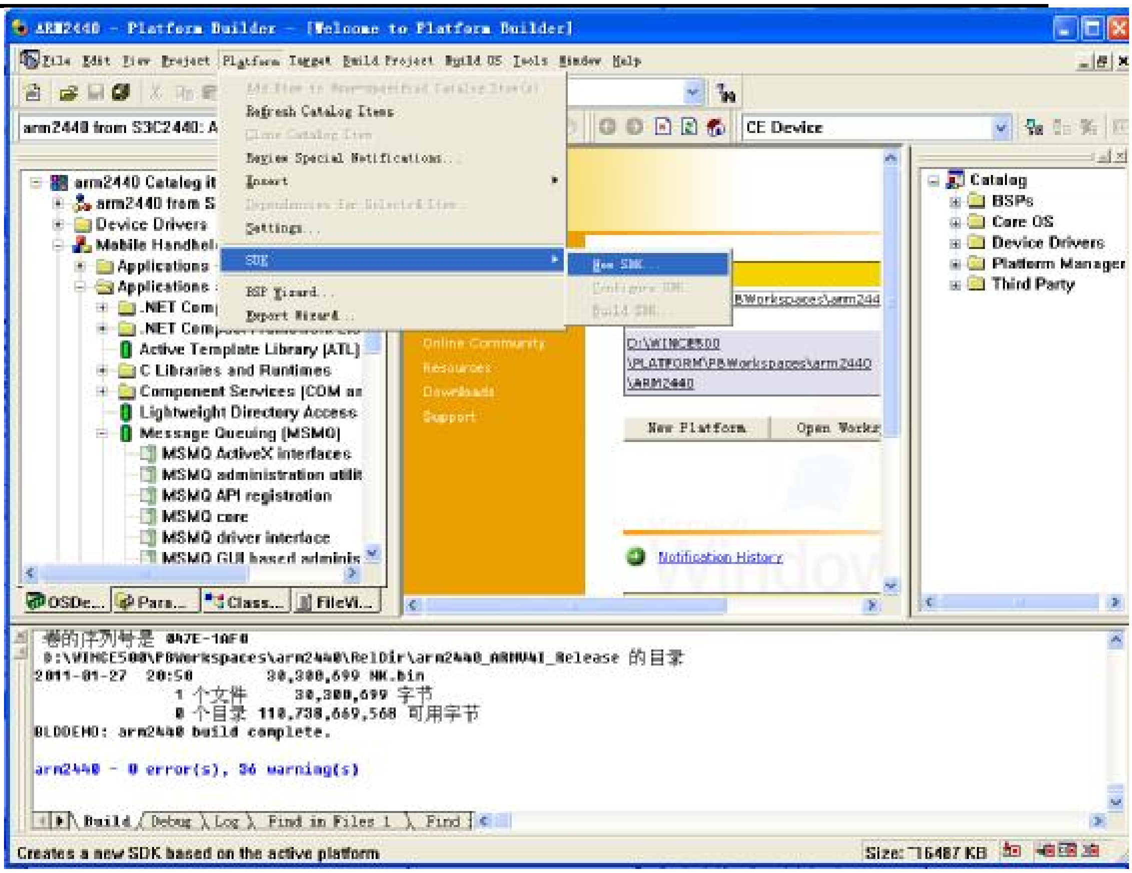
9.1.5 导出 SDK

我们可以把定制好的内核工程导出为SDK 安装文件，它用来提供给应用开发人员，里面主要包含与定制平台有关的头文件、库、一些文档等内容。应用开发人员可以通过安装 SDK 在Embedded Visual C++(以下简称EVC)中开发基于此平台的应用程序。

说明：光盘**\WindowsCE5.0\SDK**目录中有已经制作好的SDK 安装文件，您可以直接使用而不必自己制作。

下面是具体的导出步骤。

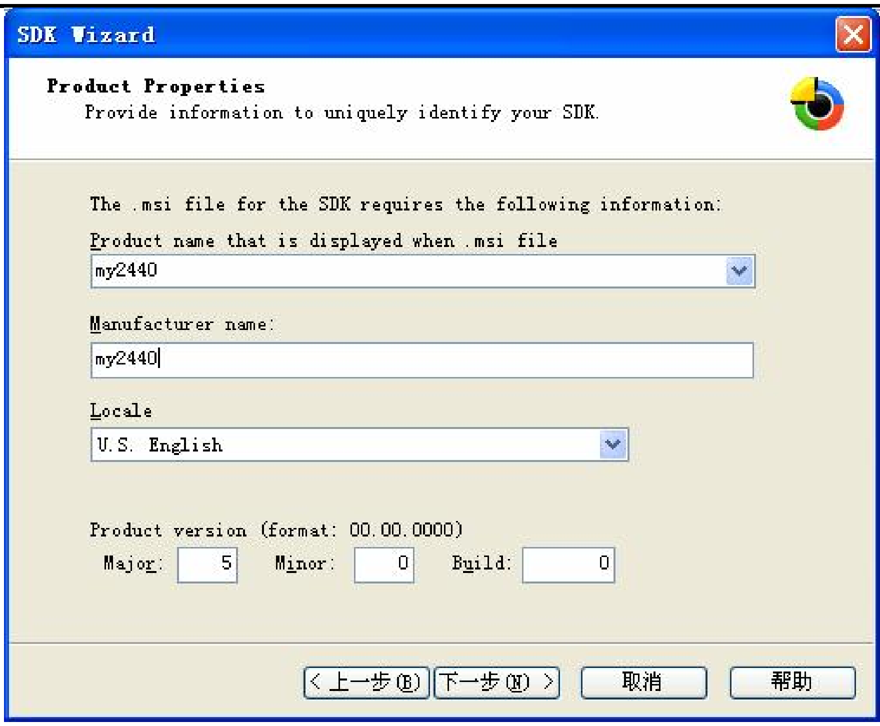
Step1: 首先打开并确定已经编译好工程示例，点【Platform】|【SDK】|【New SDK…】如图：



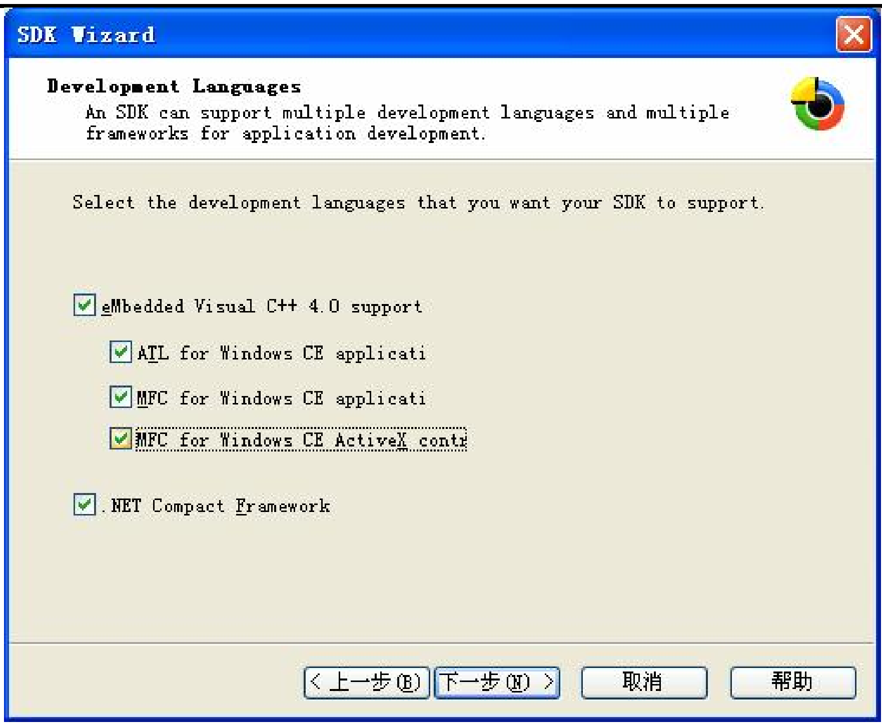
Step2: 跳出“Export SDK Wizard”向导窗口，点“下一步”继续：



Step3: 进入“Prodect Properties”配置窗口，可以根据实际情况填写配置，点“下一步”继续，如图：



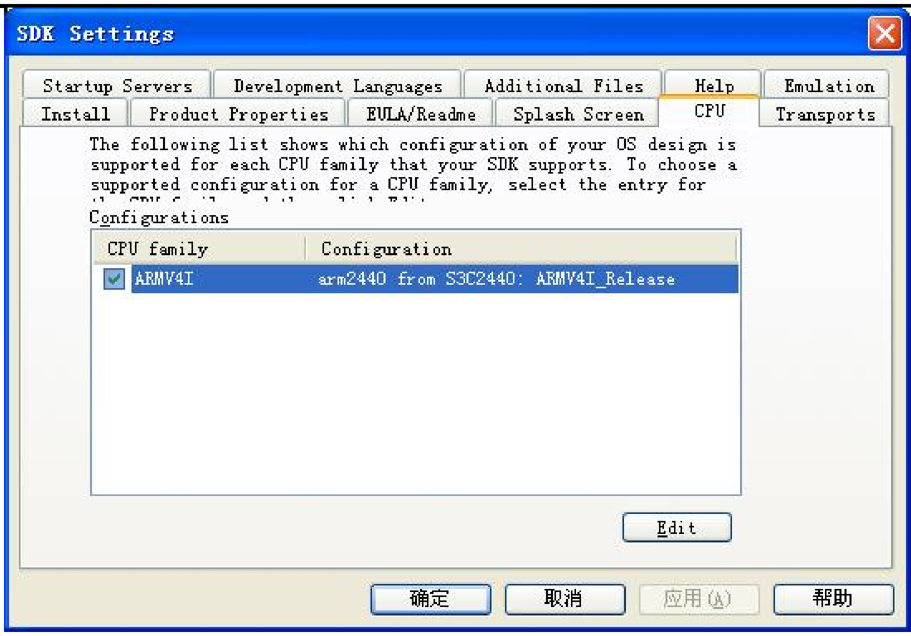
Step4: 进入“Development Language”配置窗口，选择开发语言支持，点“下一步“:



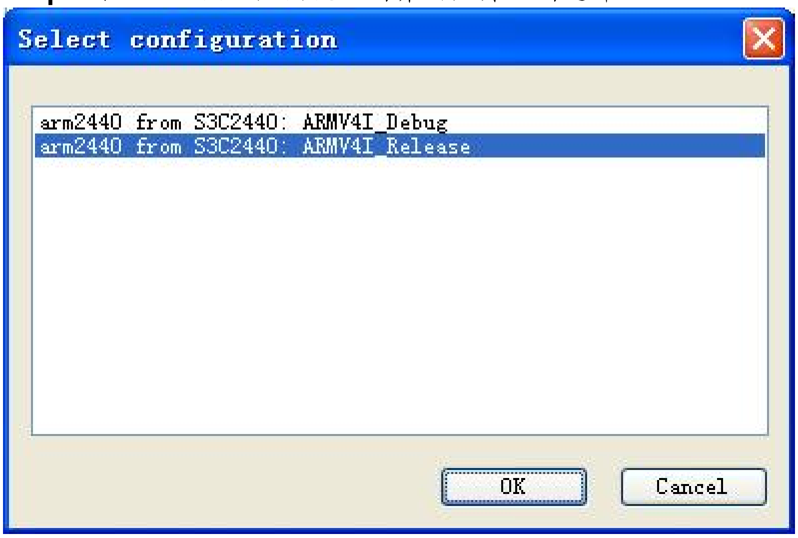
Step5: 配置完毕，点“Fnish“按钮结束。



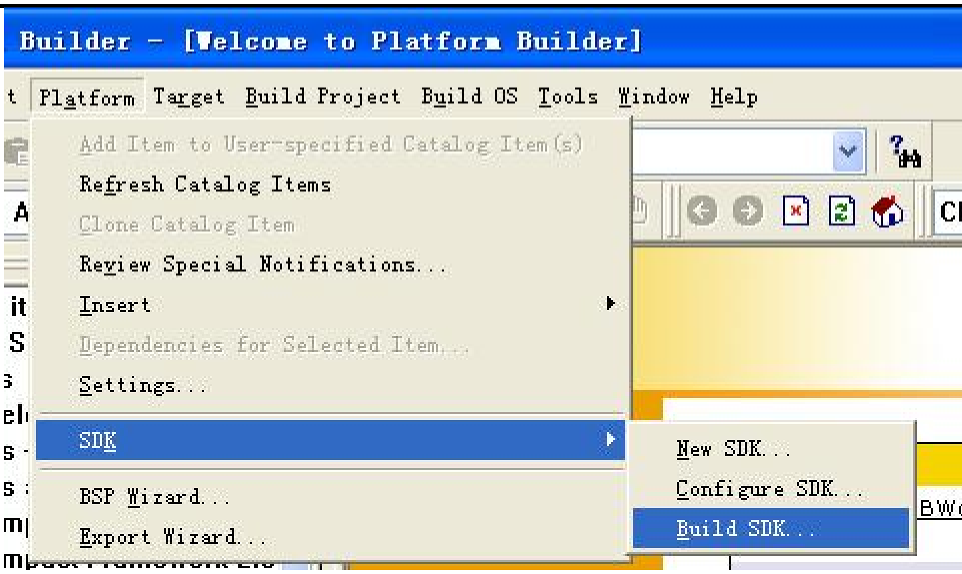
Step6: 点【Platform】|【SDK】|【Configure SDK…】，出现设置窗口，在这里你可以对刚才的初始配置进行更加详细的设置，点"CPU"选项卡，出现如图界面：



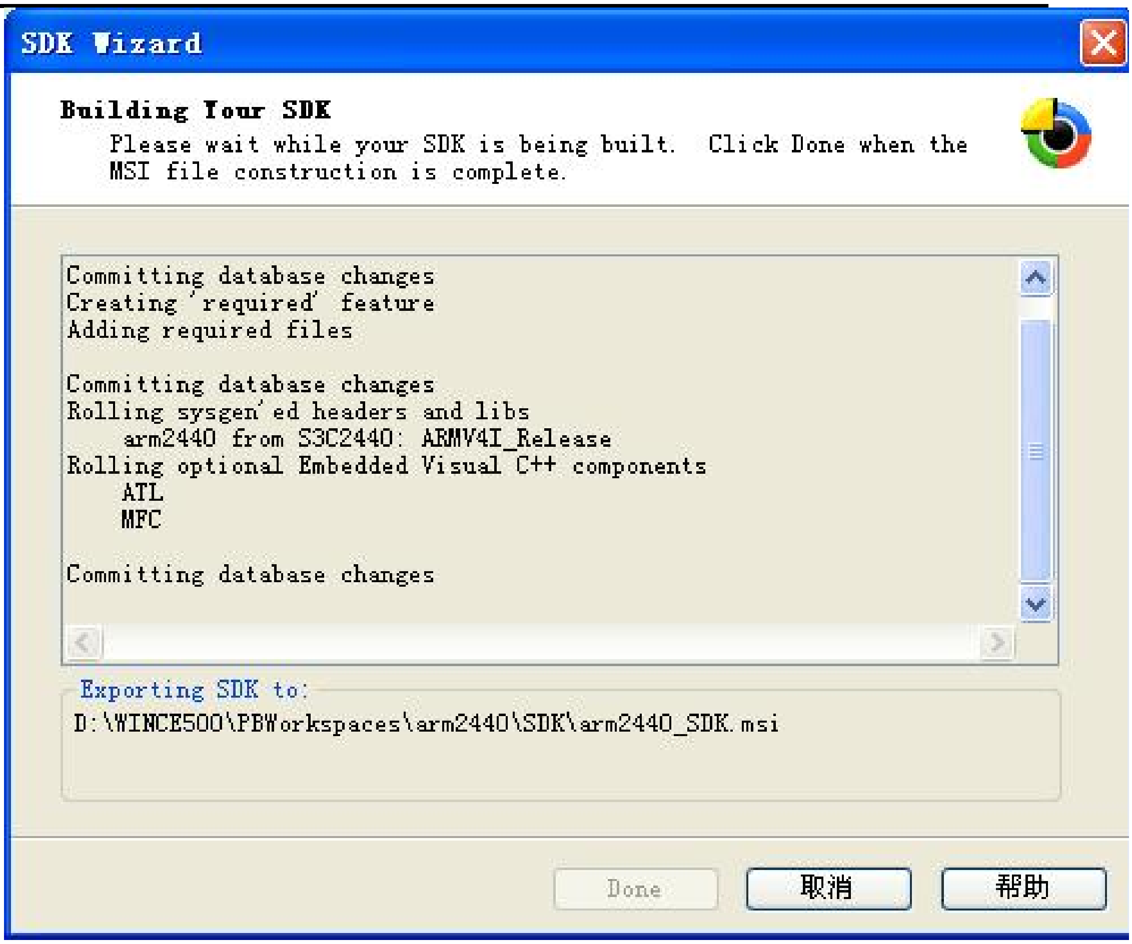
Step7: 点"Edit"按钮，出现如图界面，并如图选择:



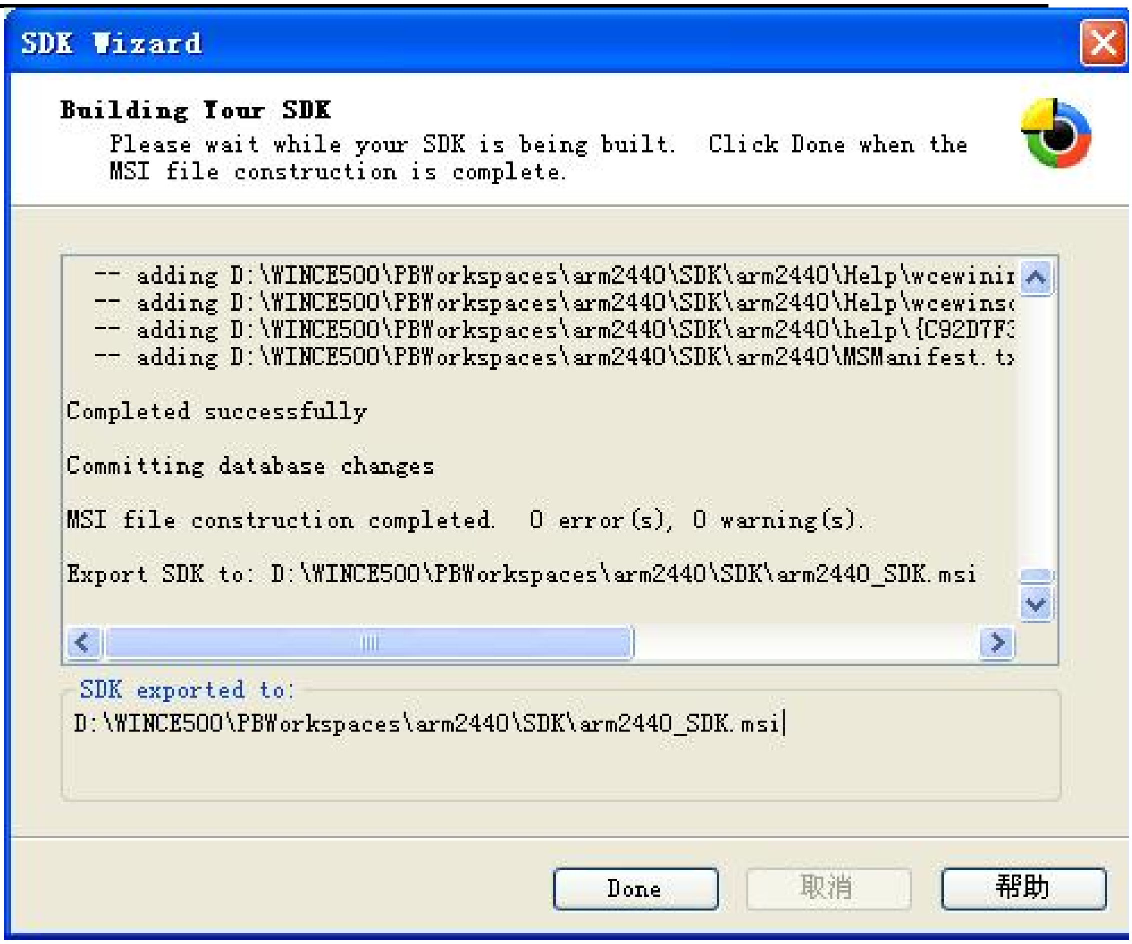
Step8: 点"OK"返回PB5 主界面，再点【Platform】|【SDK】|【Build SDK…】:



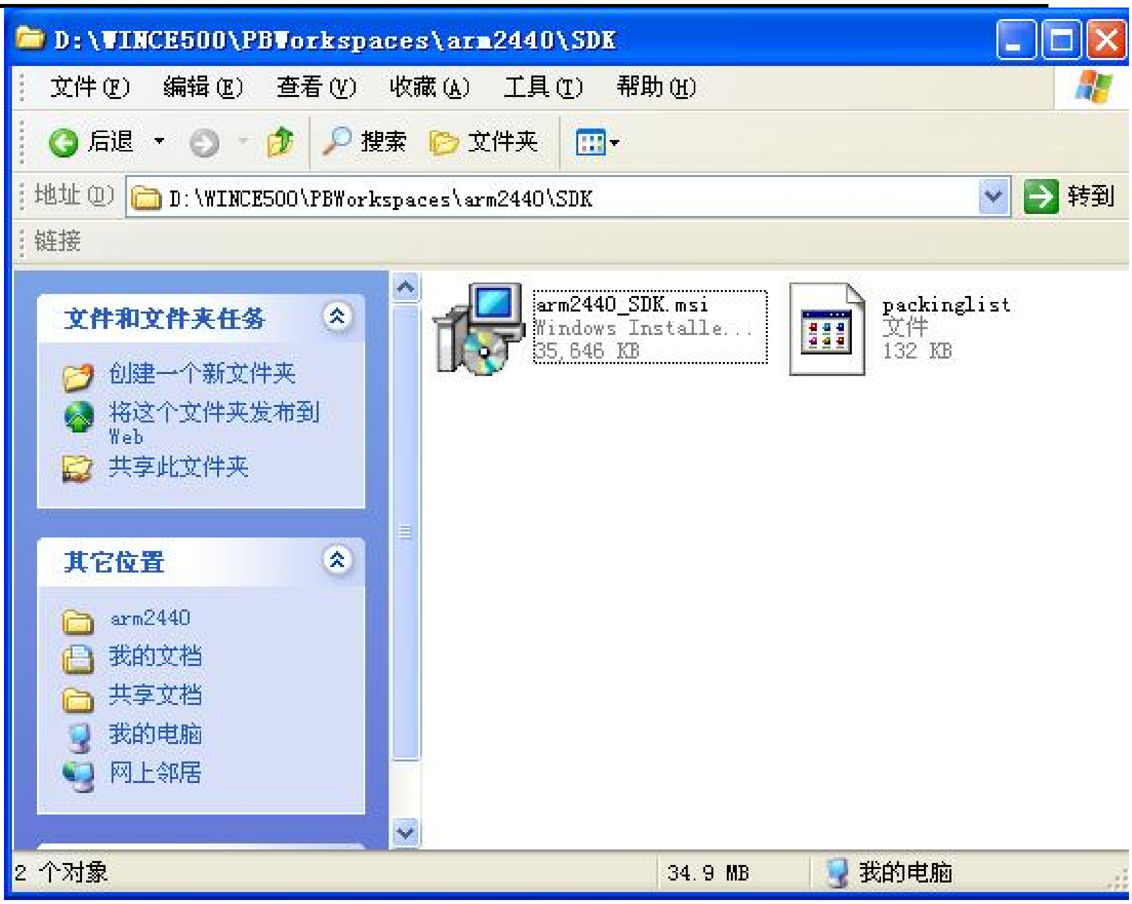
Step9: 出现编译向导窗口，并同时开始编译制作SDK:



Step10: 大概几分钟时间，编译完毕，此时点“Done“按钮结束：



Step11: 根据提示，最后在如图目录生成SDK 安装文件:



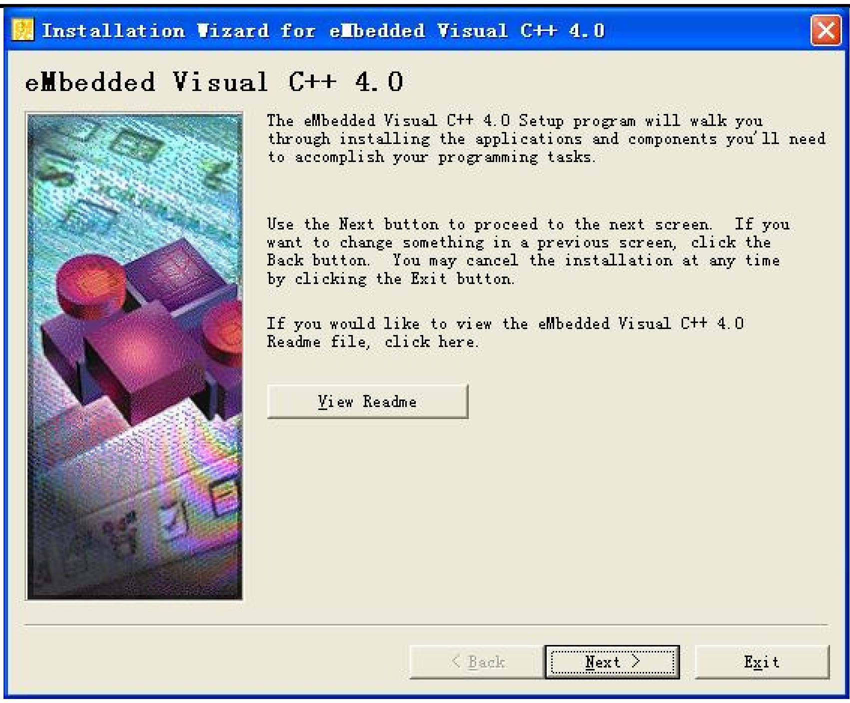
9.1.6 安装 Embedded Visual C++(EVC)

为了开发基于API 的WinCE 应用程序，需要安装EVC 集成开发环境和相应的SDK及补丁，下面是详细的EVC 安装步骤：

(1)EVC 安装文件位于光盘**WinCE源代码和软件\WindowsCE5.0\Embedded VisualC++**目录中，双击setup.exe 开始安装



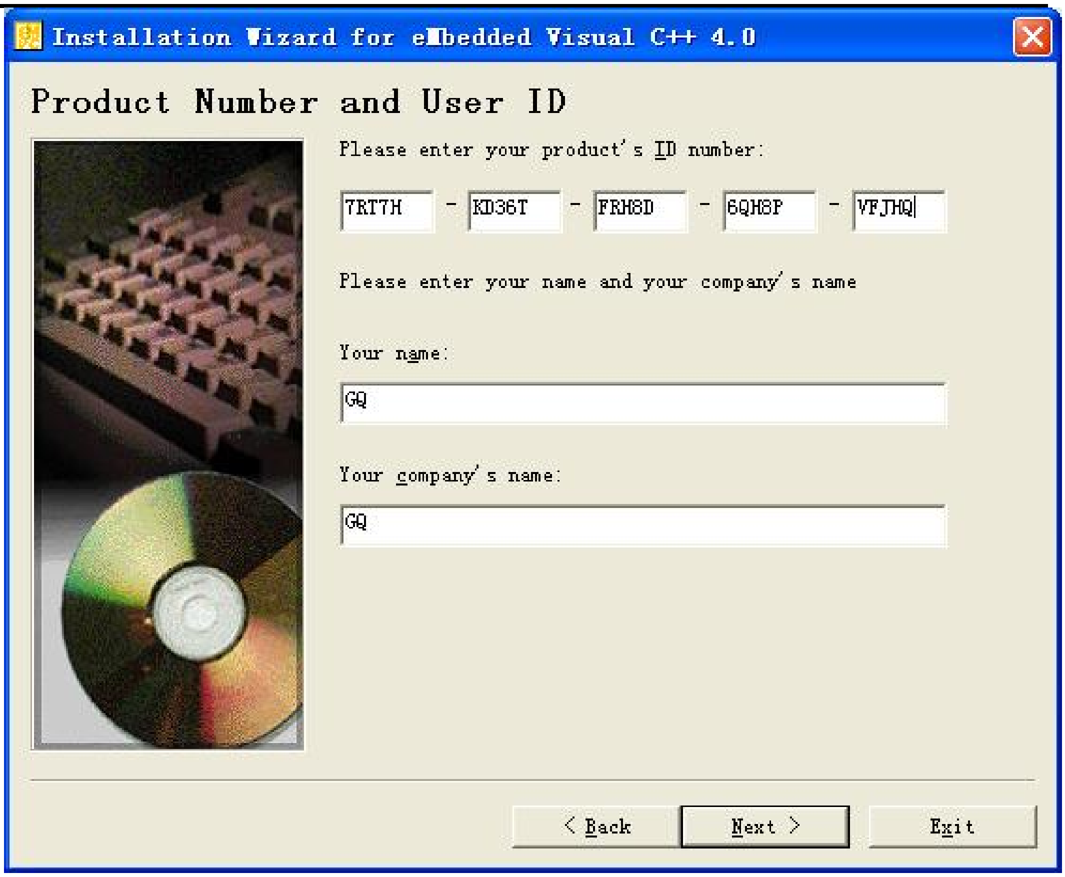
(2)出现安装向导界面，点“Next”继续



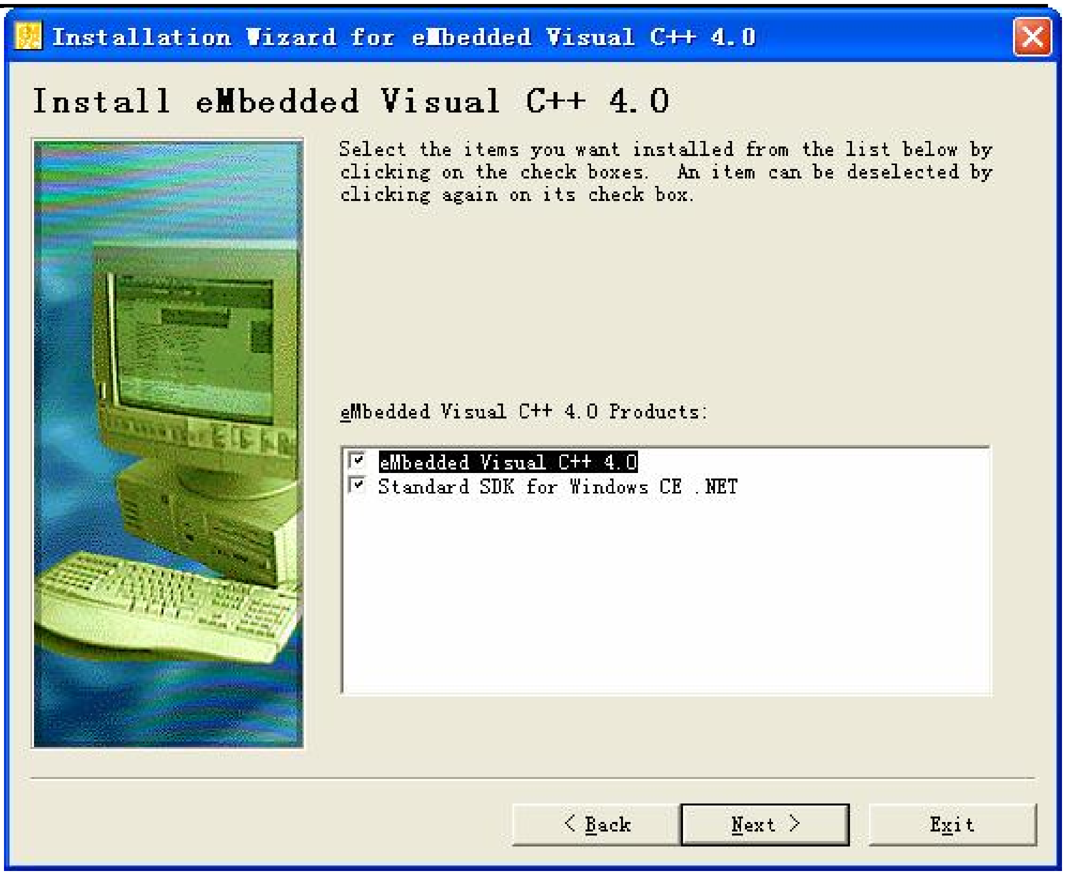
(3)出现用户许可协议，选择“I accept the agreement”点“Next”继续



(4)输入序列号和用户信息，点“Next”继续



(5)选择安装组件，选择默认全部安装即可，点“Next”继续



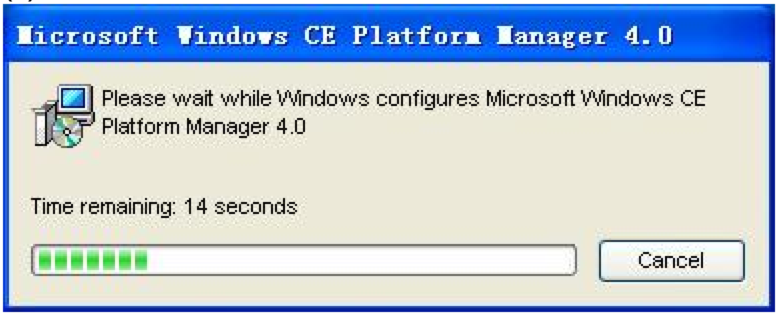
(6)选择安装目录，这里按默认，点“Next”继续



(7)跳出提示窗口，选择“是”继续



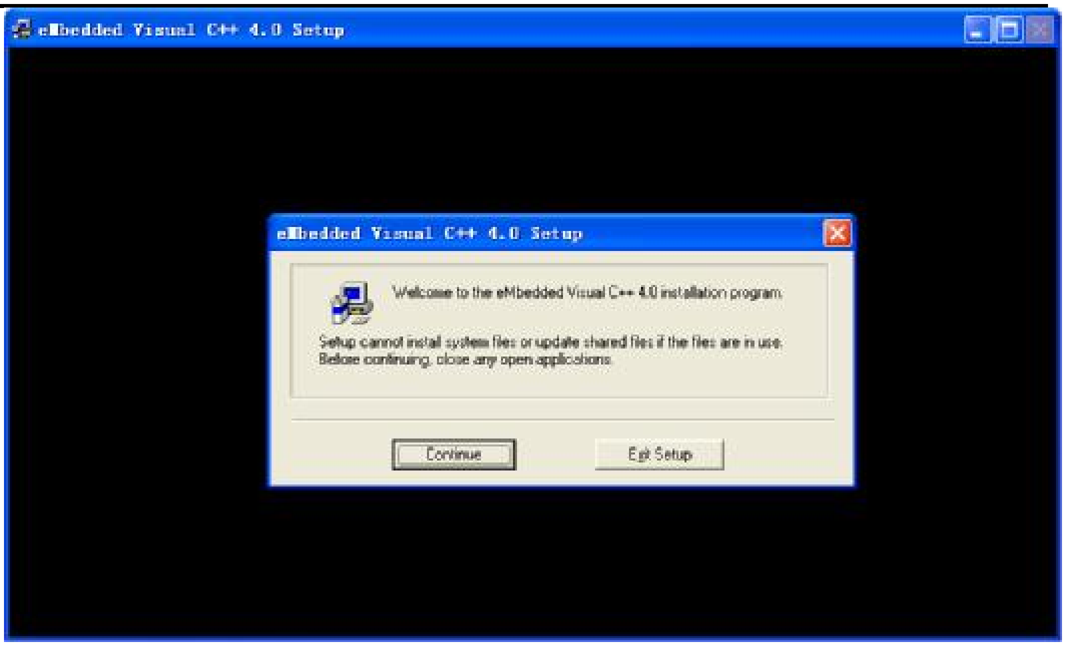
(8)开始安装EVC 管理器，出现如图过程窗口，等待即可



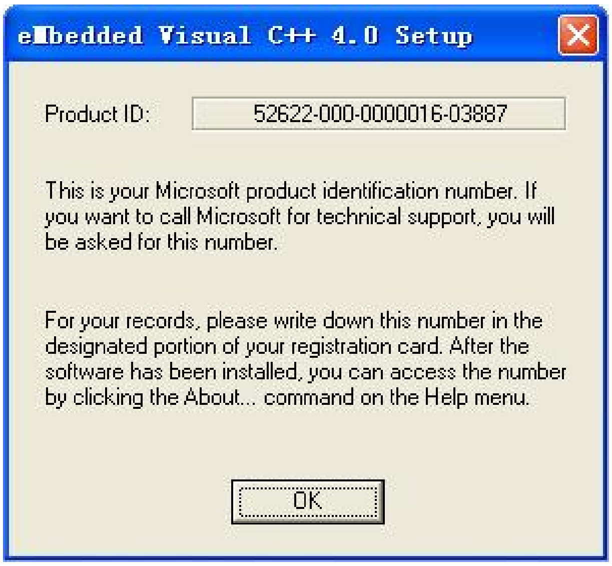
(9)EVC 平台管理器安装完毕，点“OK”结束



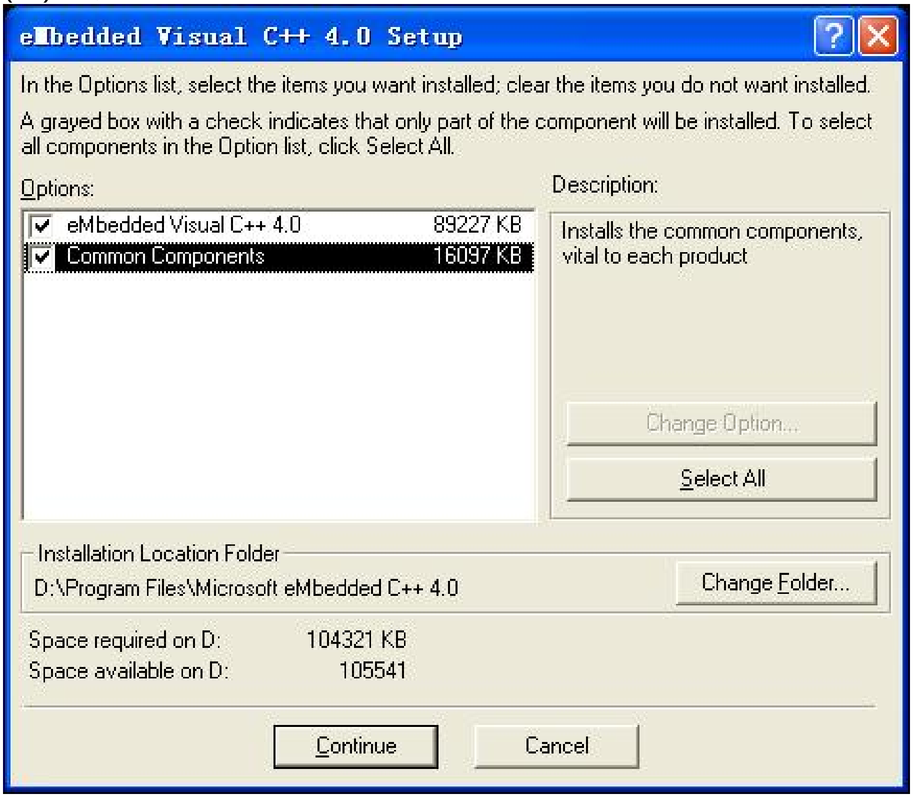
(10)出现EVC 安装界面，点“Continue”继续



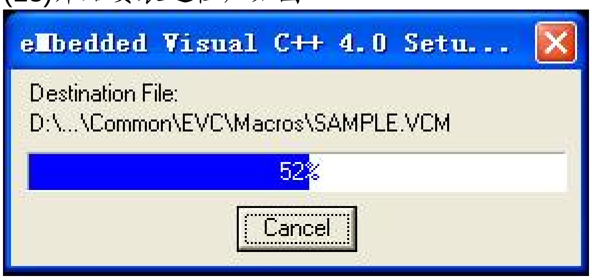
(11)出现产品ID 号提示信息窗口，点“OK”继续



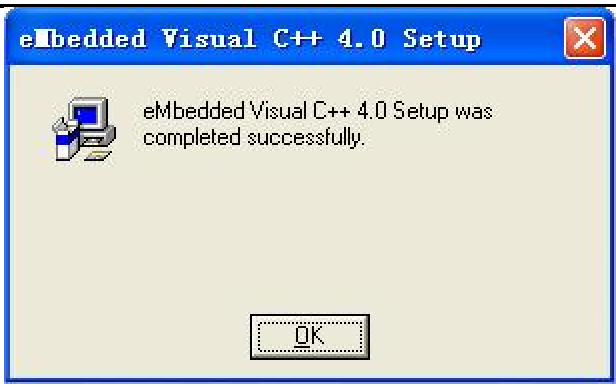
(12)选择安装组件和安装路径，这里把盘符改为D，点“Next”继续



(13)开始安装进程，如图



(14)安装完毕，点“OK”结束



注意：此时如果点【开始】|【程序】|【Microsoft eMbedded Visual C++ 4.0】|【eMbedded Visual C++ 4.0】快捷菜单，尚不能正常运行，会出现以下提示窗口，因此你还需要按照下面的章节继续安装SDK才可以。



9.1.7 安装 EVC 补丁和导出的 SDK

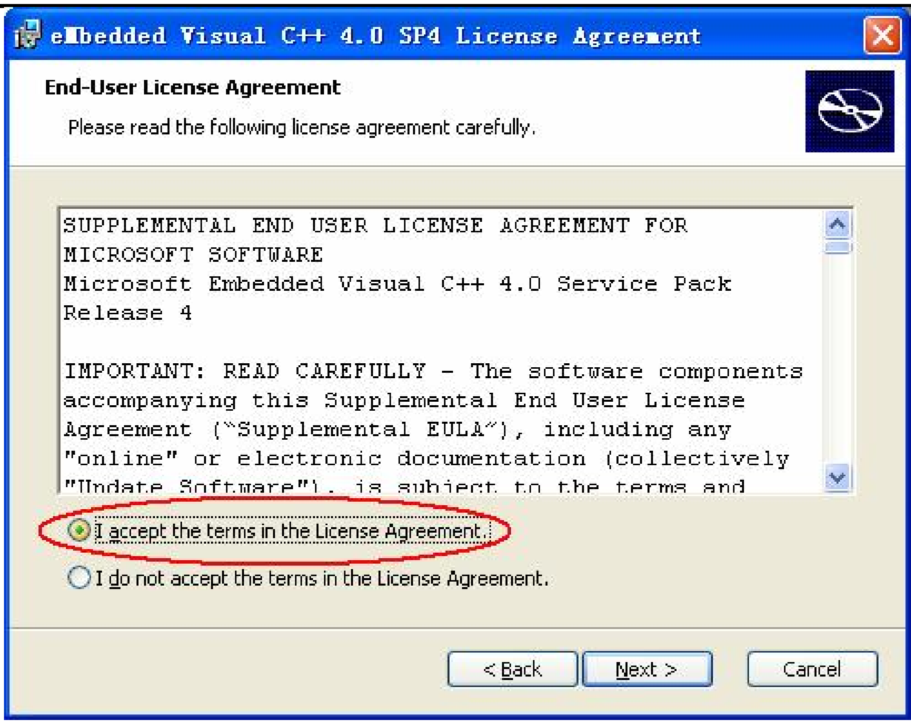
为了能够正常使用我们导出的SDK 安装文件，必须要先安装EVC 的SP4 补丁文件，它位于**\Embedded VisualC++\SP\evc4sp4\DISK1**目录中，下面的步骤我们先安装补丁文件，再安装刚才我们导出的SDK，如图：



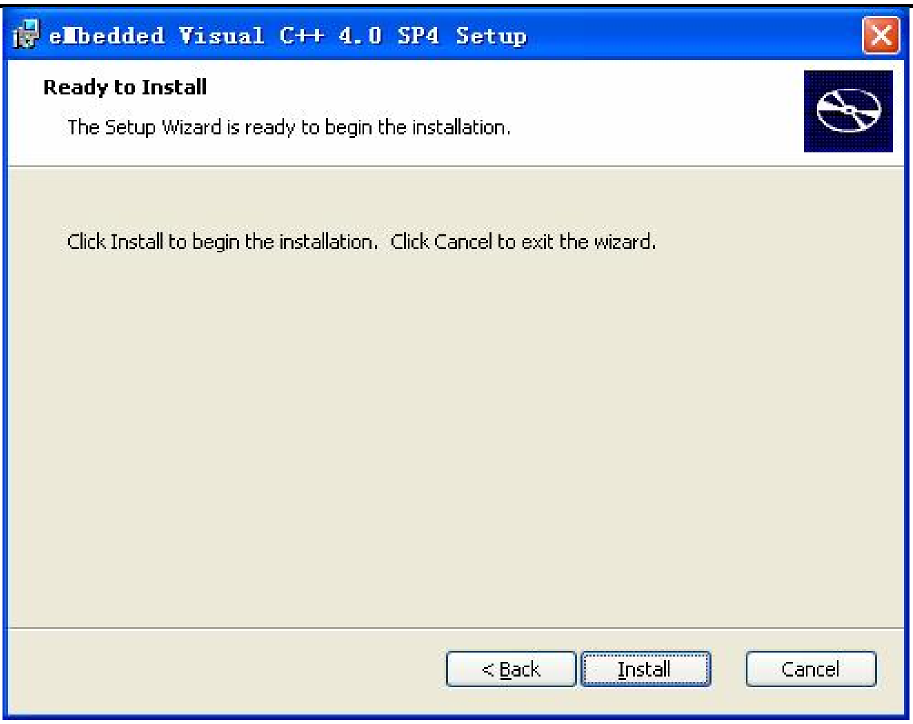
(1)双击运行SP4 的安装程序setup，出现安装向导界面，点“Next”继续



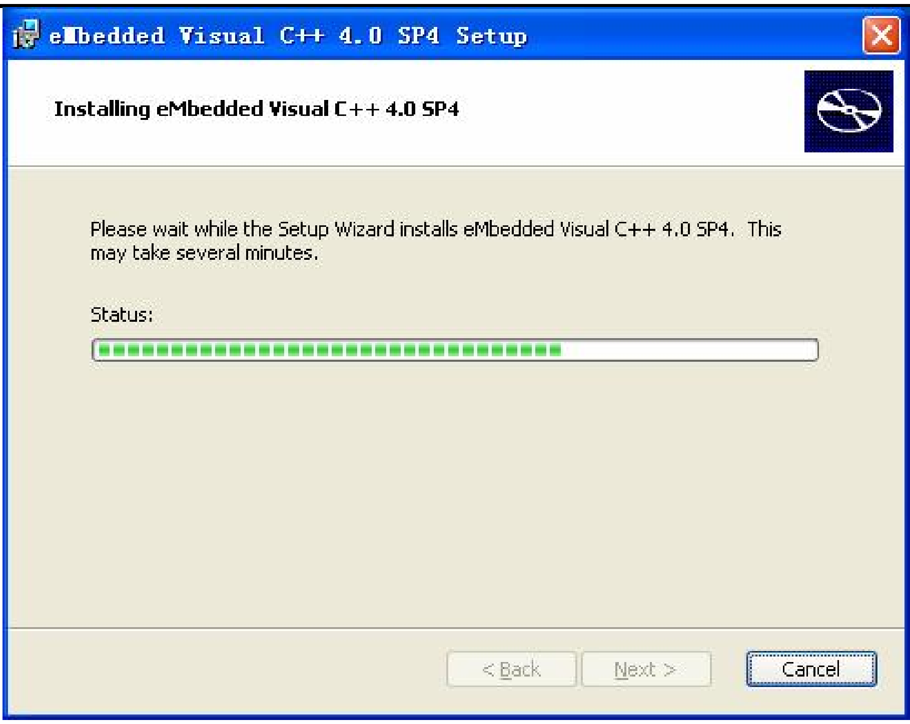
(2)出现用户许可协议窗口，如图选择，点“Next”继续



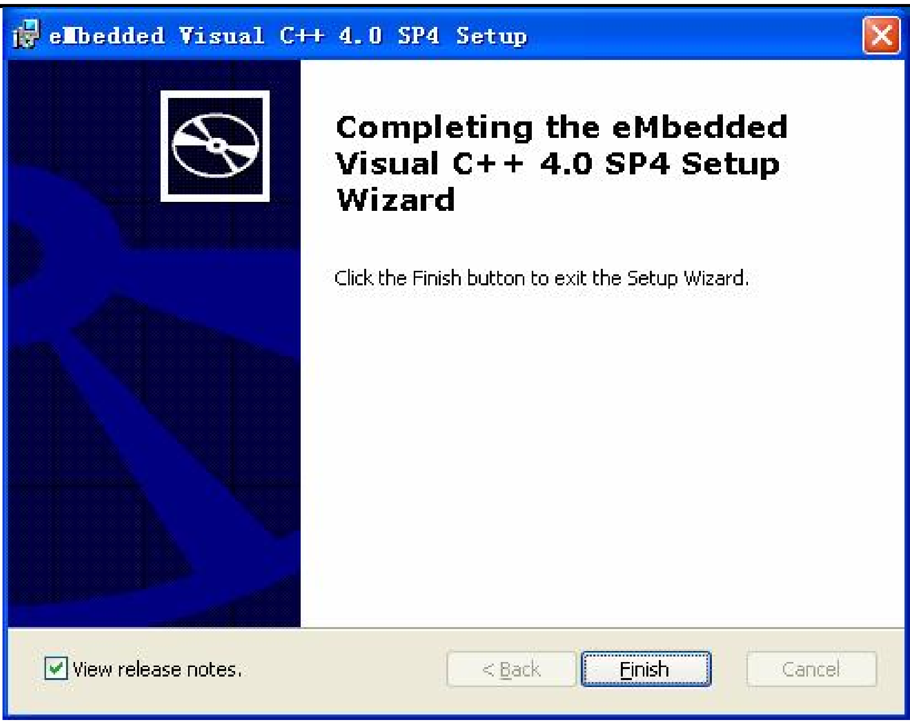
(3)准备安装SP4，点“Next”继续


eMbedded Visual C++ 4.0 SP4 Setup
Ready to Install
The Setup Wizard is ready to begin the installation.
Click Install to begin the installation. Click Cancel to exit the wizard.
< Back
Install
Cancel

(4)开始安装进程，如图


eMbedded Visual C++ 4.0 SP4 Setup
Installing eMbedded Visual C++ 4.0 SP4
Please wait while the Setup Wizard installs eMbedded Visual C++ 4.0 SP4. This may take several minutes.
Status:
< Back
Next >
Cancel

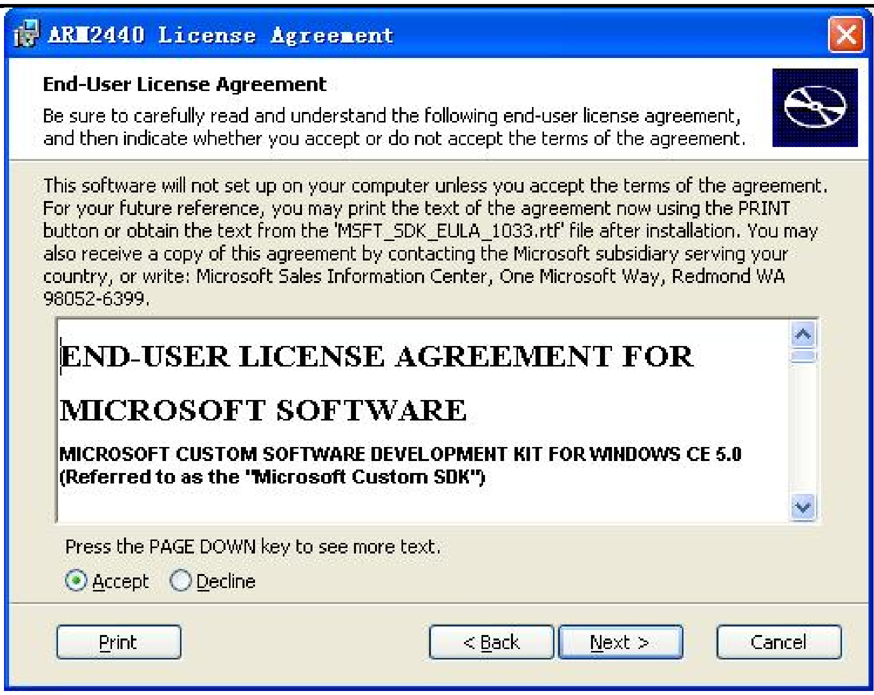
(5)安装结束，如图



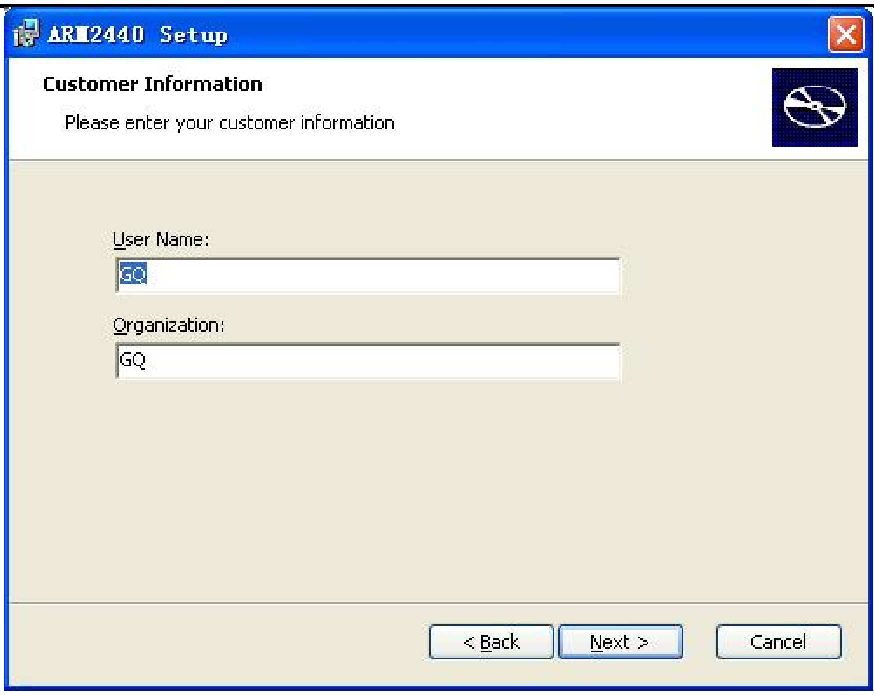
(6)现在我们开始安装刚才导出的SDK，你可以使用自己导出的，也可以在光盘中找到我们已经做好的(它位于光盘的WindowsCE5.0\SDK 文件夹中， 名字为"GQ2440_SDK.msi.exe")，双击运行它，出现安装向导窗口，点“Next”继续：



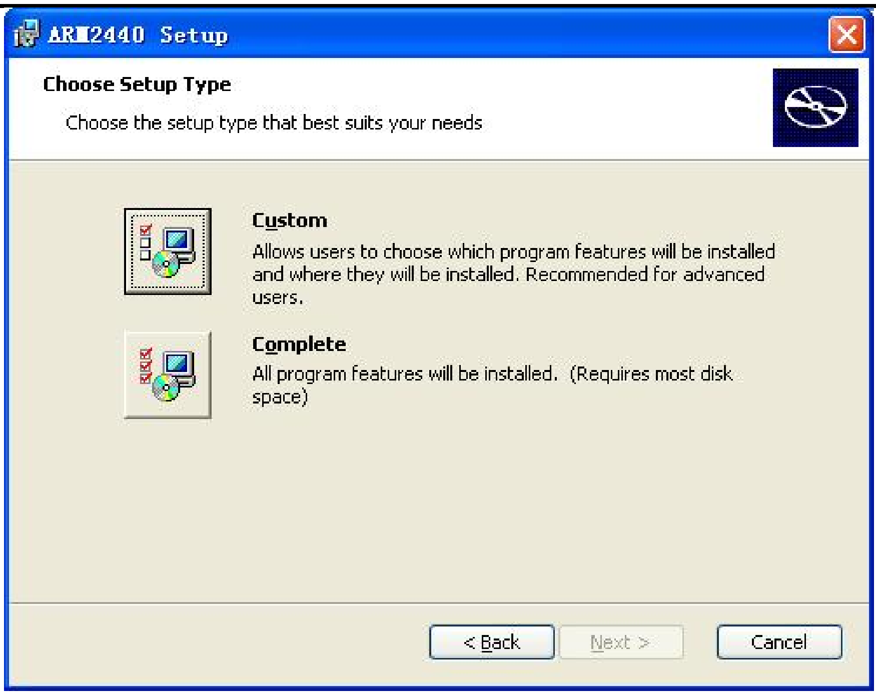
(7)出现用户许可协议窗口，选择“Accpet”，点“Next”继续：



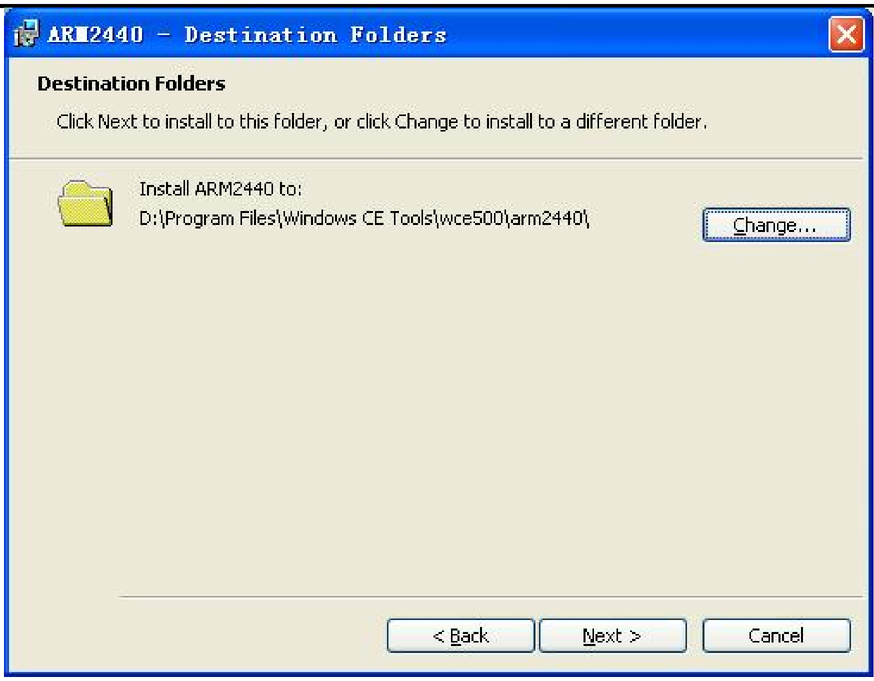
(8)出现“Customer Information”窗口，输入相应信息，点“Next”继续：



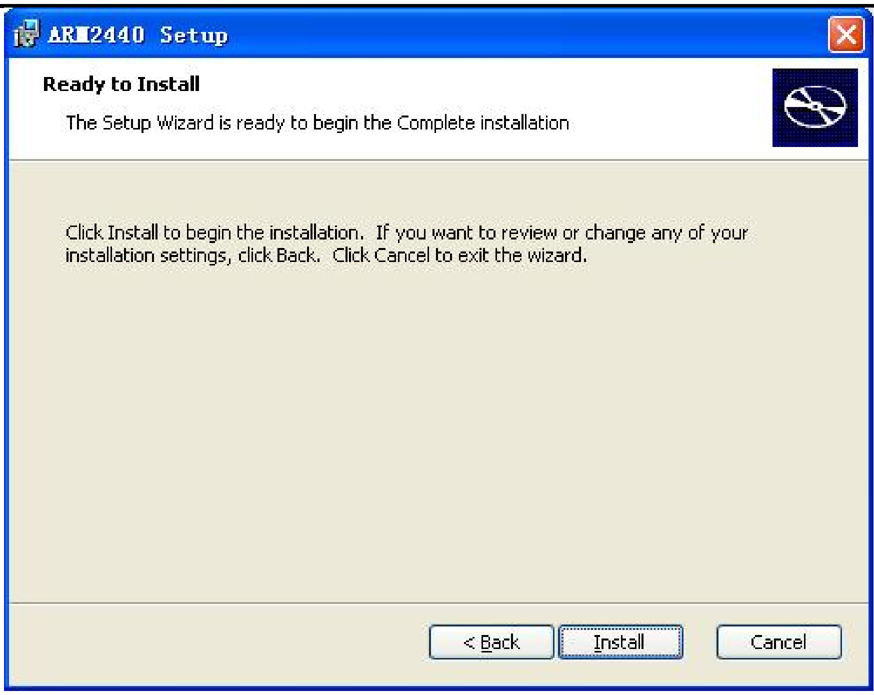
(9)出现安装类型窗口，点“Complete”完全安装继续：



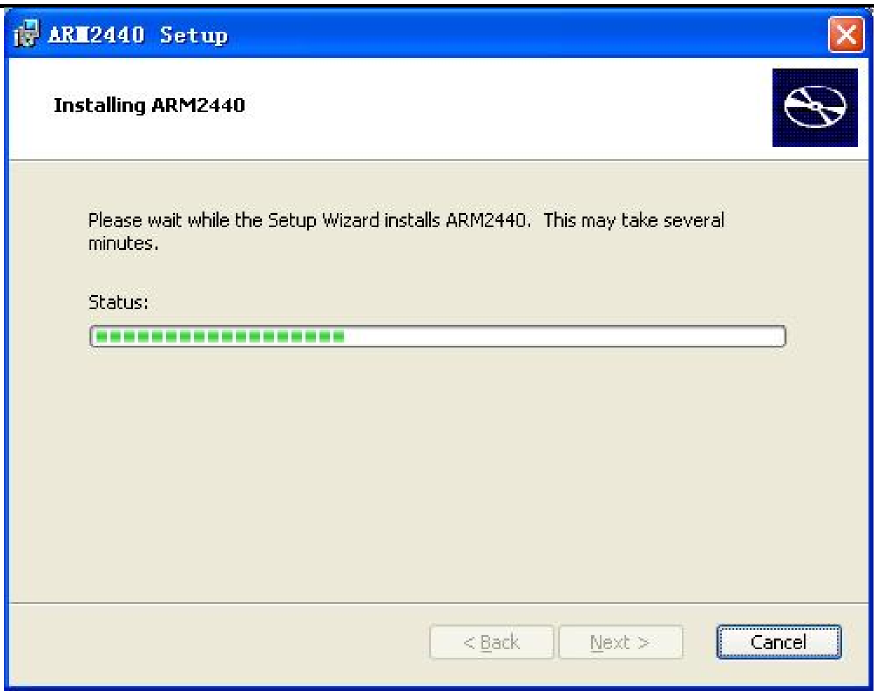
(10)出现安装目录选择窗口，这里我们将盘符改为D，点“Next”继续：



(11)在Ready to Install对话框中,,点击“Install”安装，如图:


ARM2440 Setup
Ready to Install
The Setup Wizard is ready to begin the Complete installation
Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
< Back
Install
Cancel

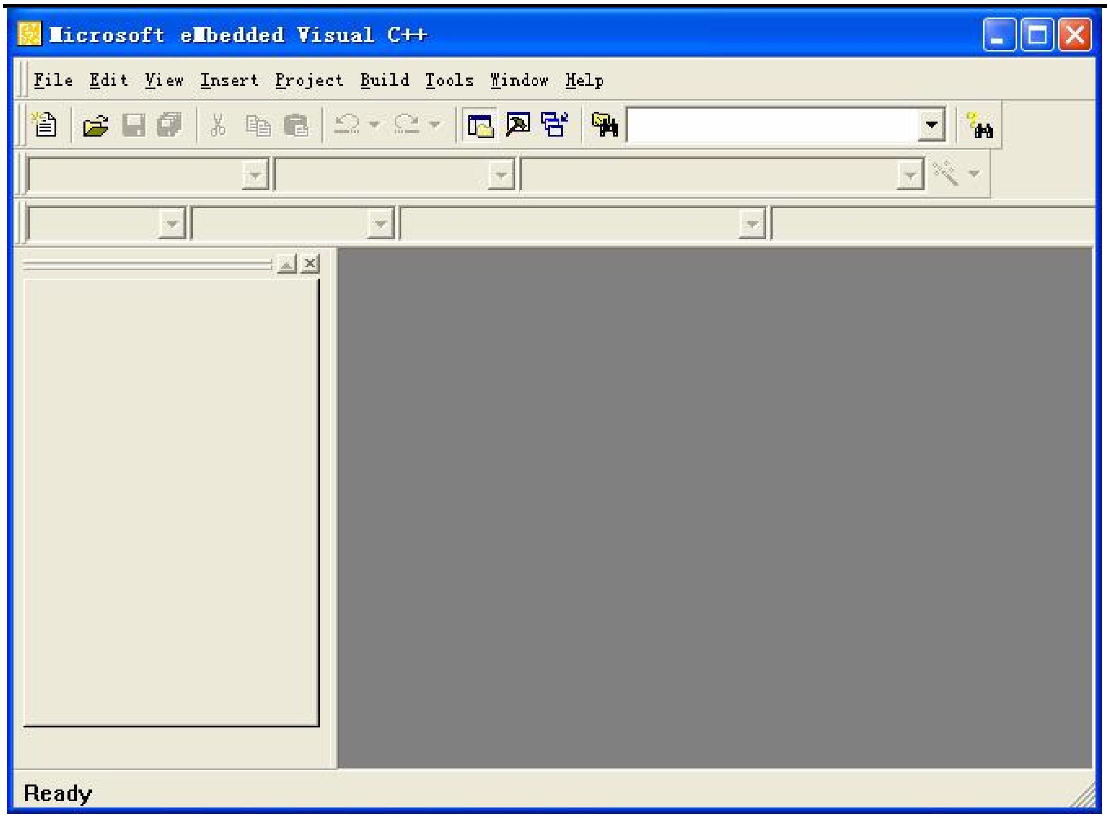
(12)系统开始安装，如图:



(13)安装完毕，点“Finish”结束:



现在，你就可以点【开始】|【程序】|【Microsoft eMbedded Visual C++ 4.0】|【eMbedded Visual C++ 4.0】快捷菜单，打开EVC 集成开发环境了，如图：

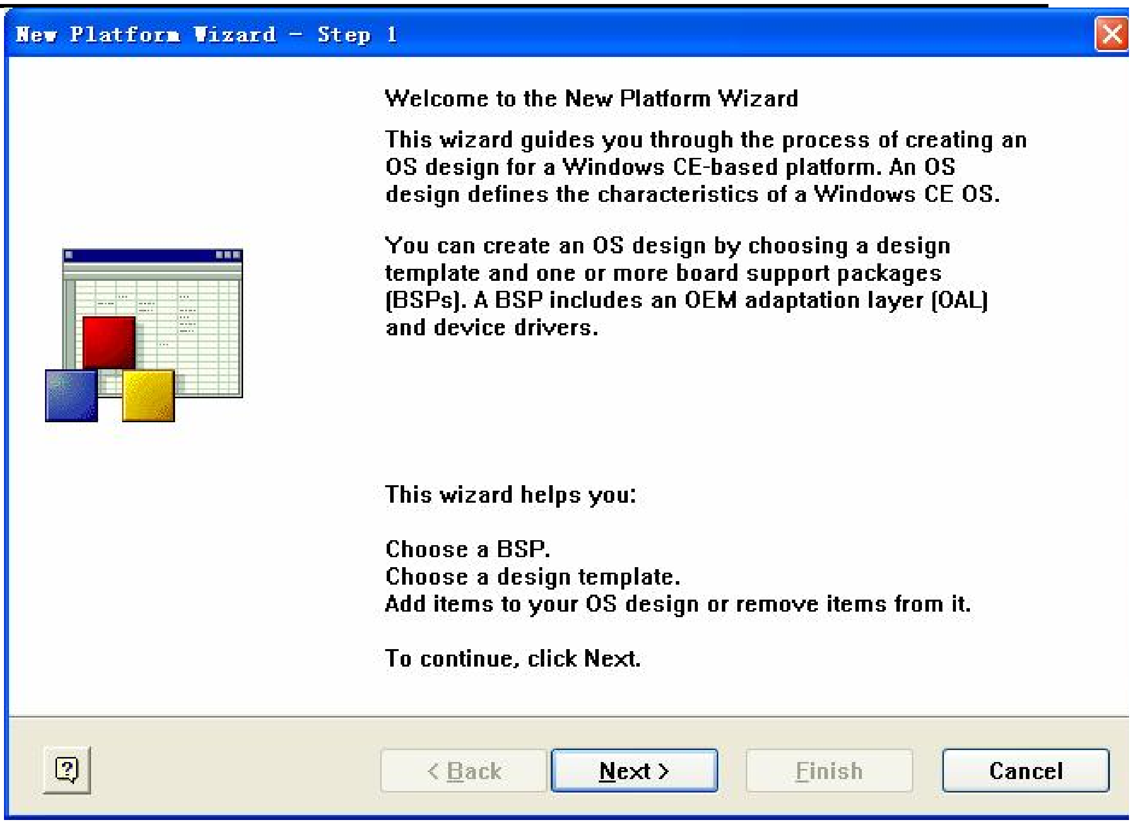


9.1.8 定制 CE 内核

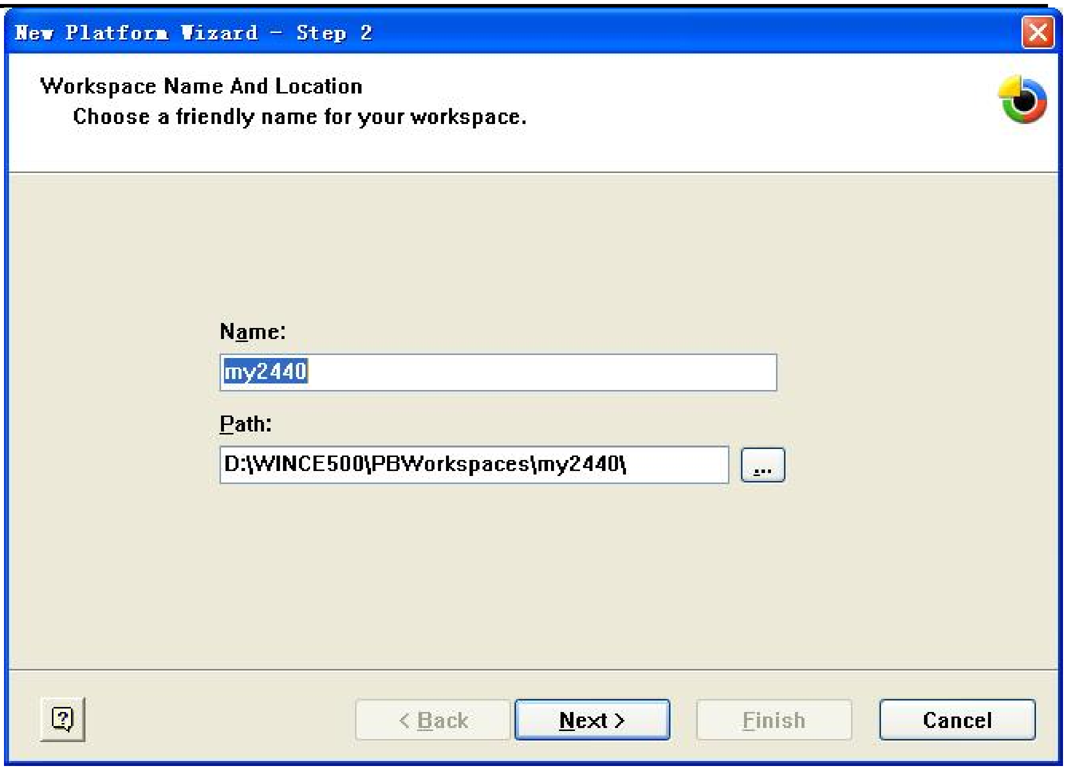
通过以上步骤，我们了解了如何安装 BSP，编译内核工程，导出SDK 等常用操作，现在我们使用图解的方式介绍一下如何定制适合于自己的WinCE 内核工程。

注意：以下步骤主要是为说明定制WinCE 内核的相关步骤和组件，根据以下步骤建立的项目不一定能顺利编译通过，请用户自行酌情增减。

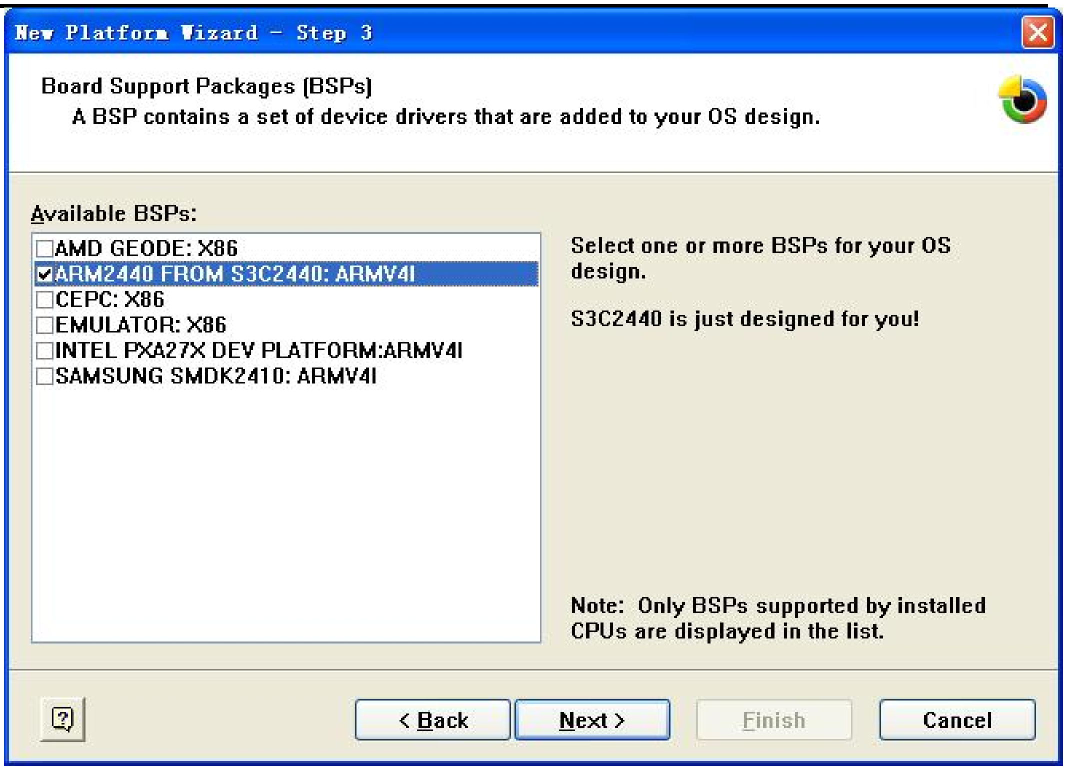
(1)打开PB5，点【File】|【New Platform…】，跳出内核定制向导，点“下一步”继续：



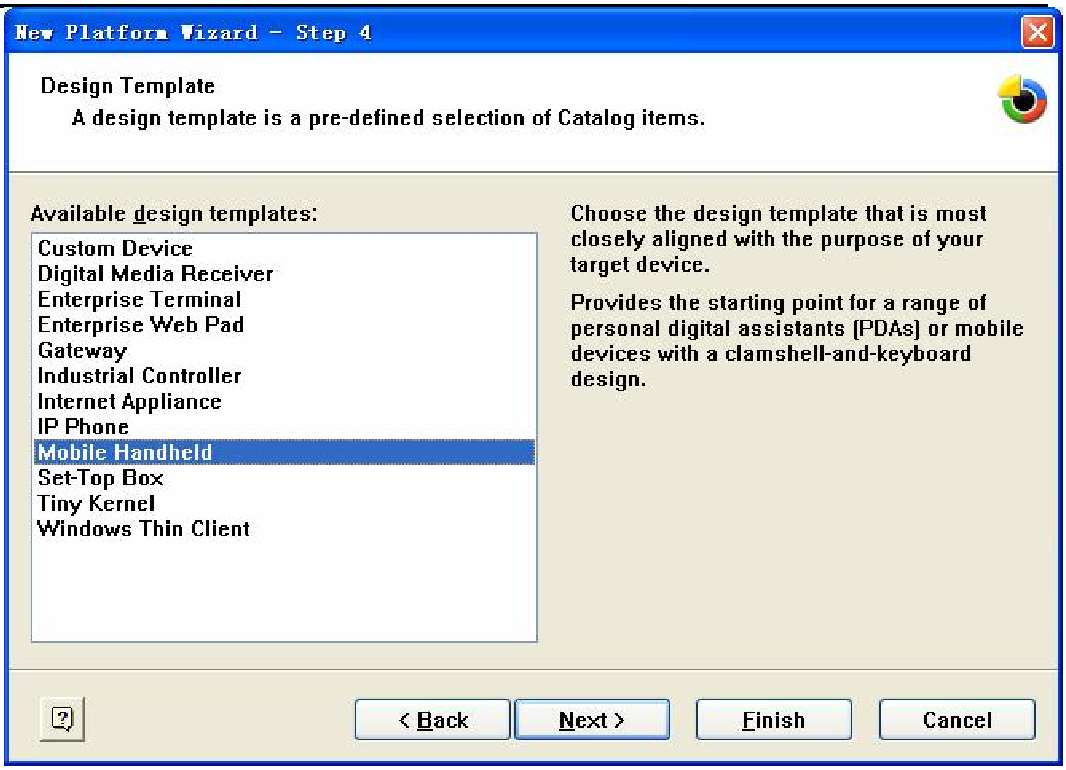
(2)出现工程命名设置窗口，输入“my2440”，点“Next”继续：



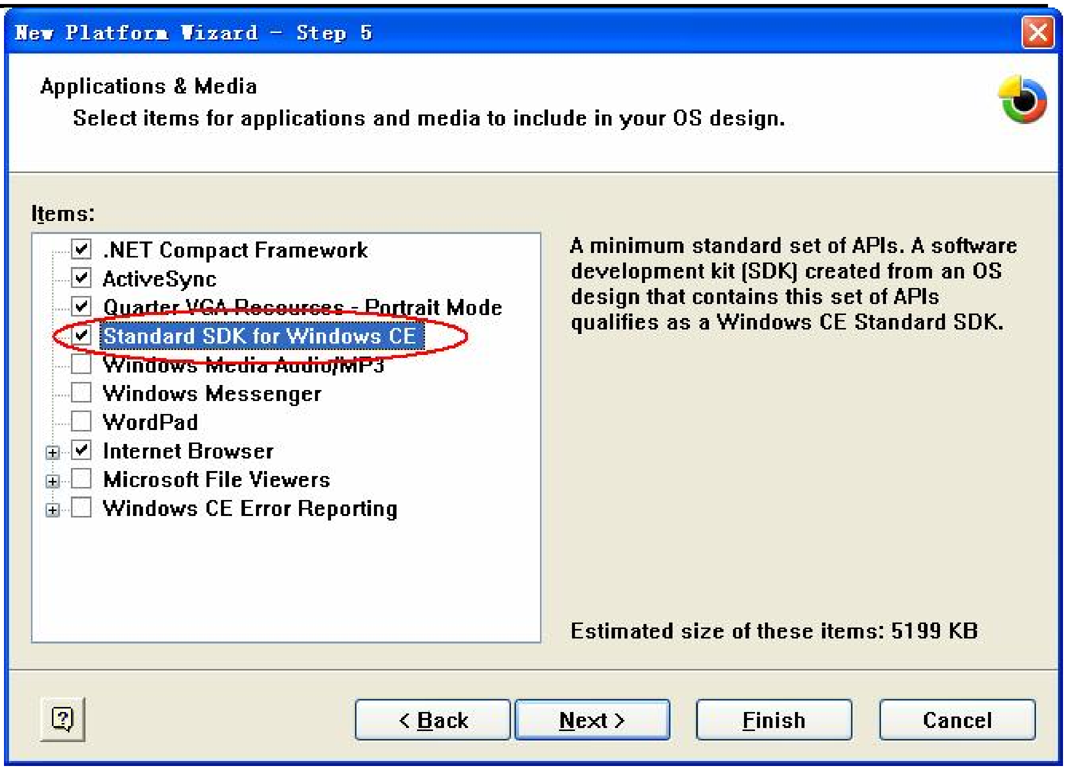
(3)选择所要使用的BSP，在这里当然选择2440 的，如图，点“Next”继续：



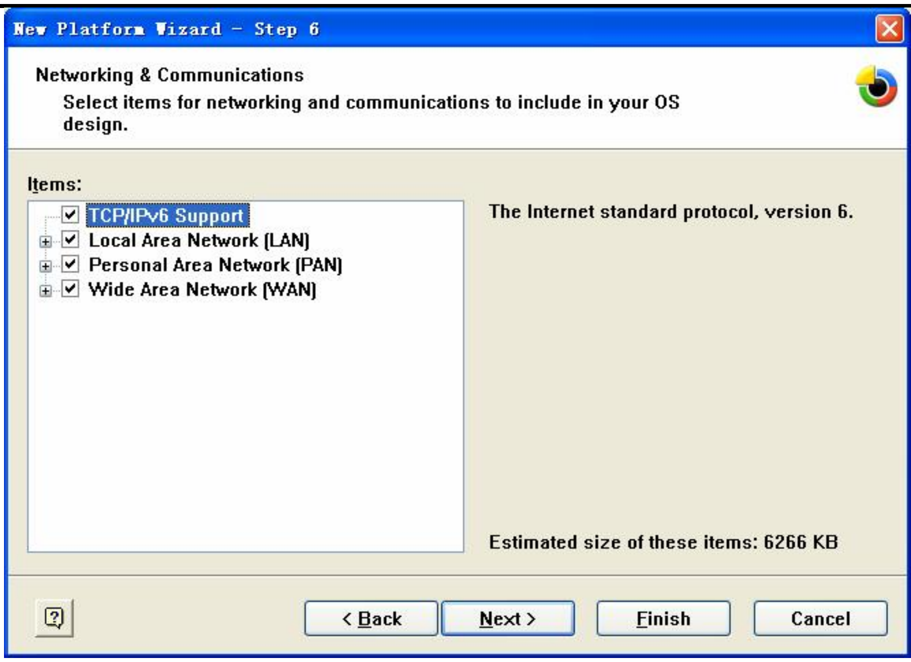
(4)出现模板选择窗口，我们选择手持设备“Mobile Handheld”，如图，点“Next”继续：



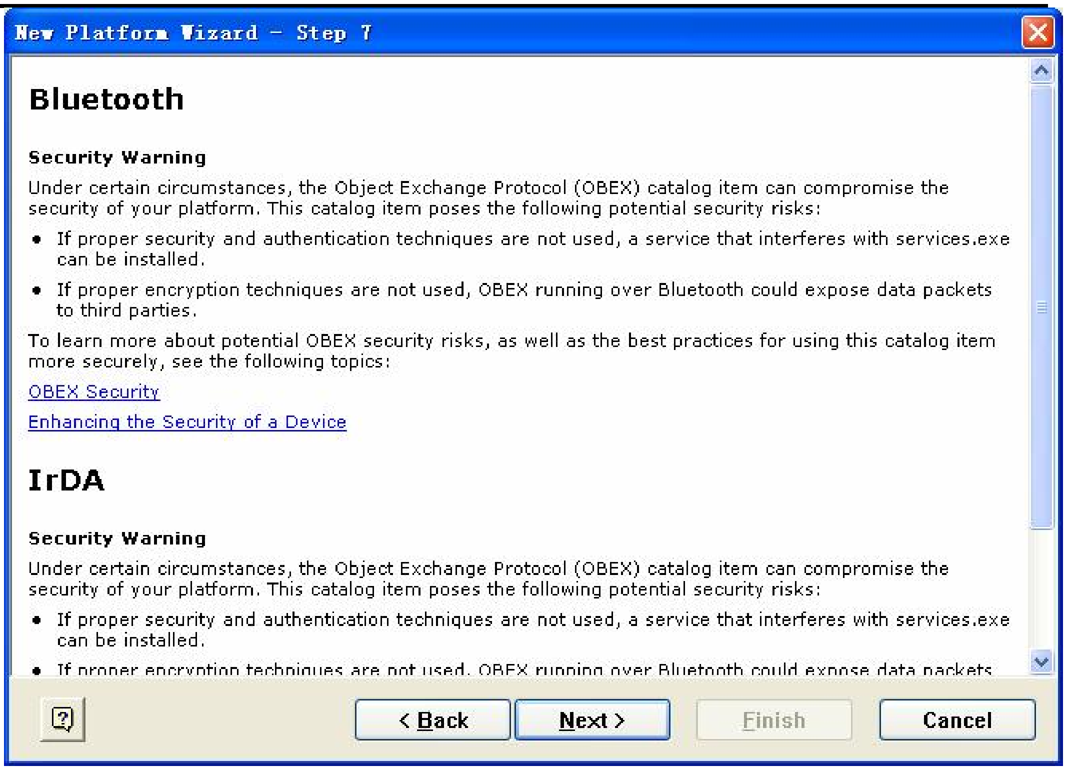
(5)出现应用程序定制窗口，这里出现的都是常见的一些应用程序，在这里**必须选择红色圈号里面的“Standard SDK for WindowsCE”**，如图，点“Next”继续：



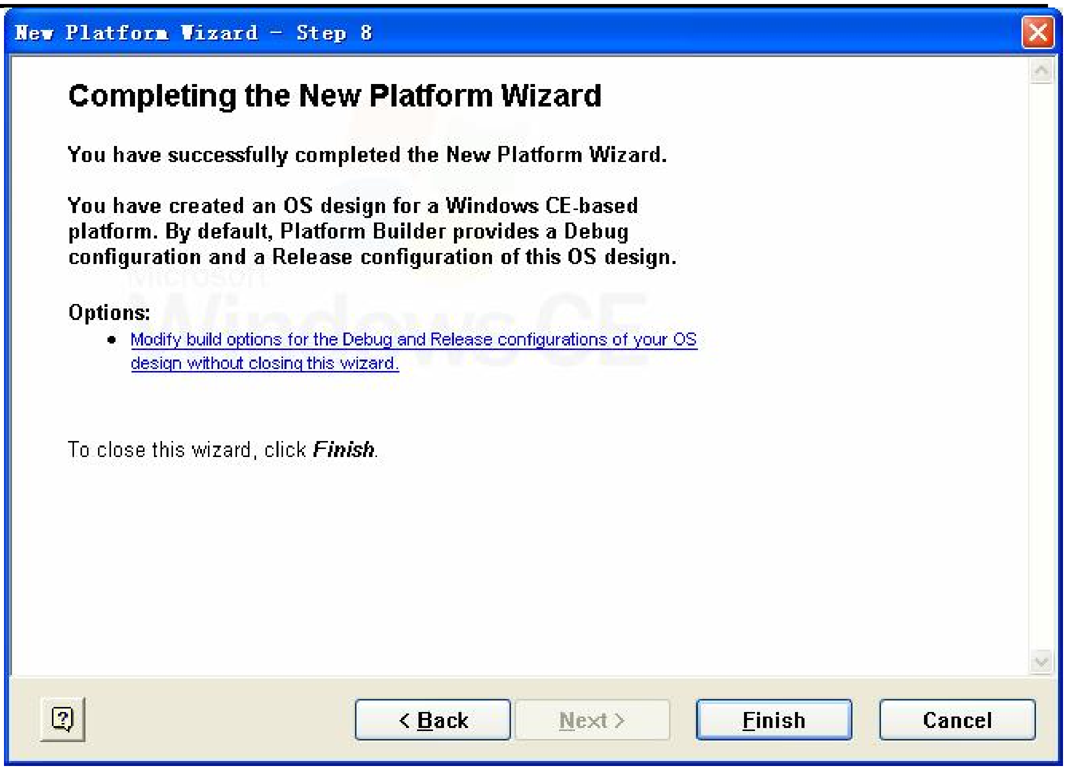
(6)出现网络有关的一些设置，使用缺省即可，点“Next”继续：



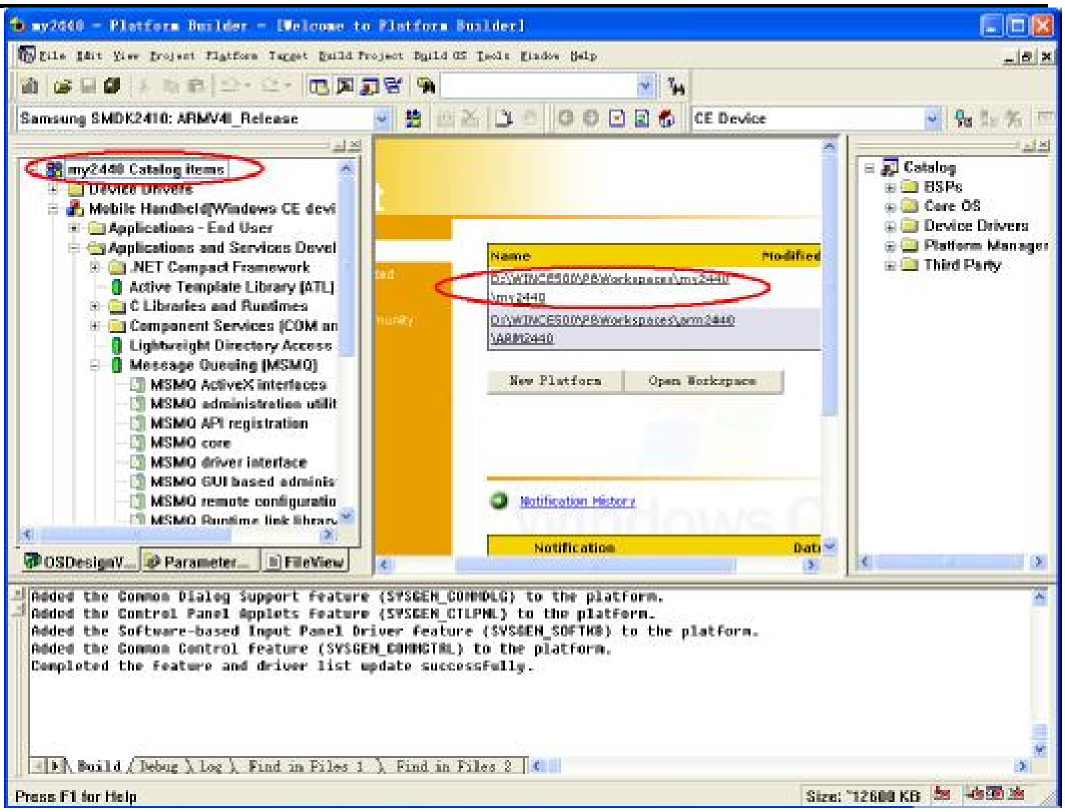
(7)出现和蓝牙、红外有关的一个窗口，这里没有任何选项，因此点“Next”继续：



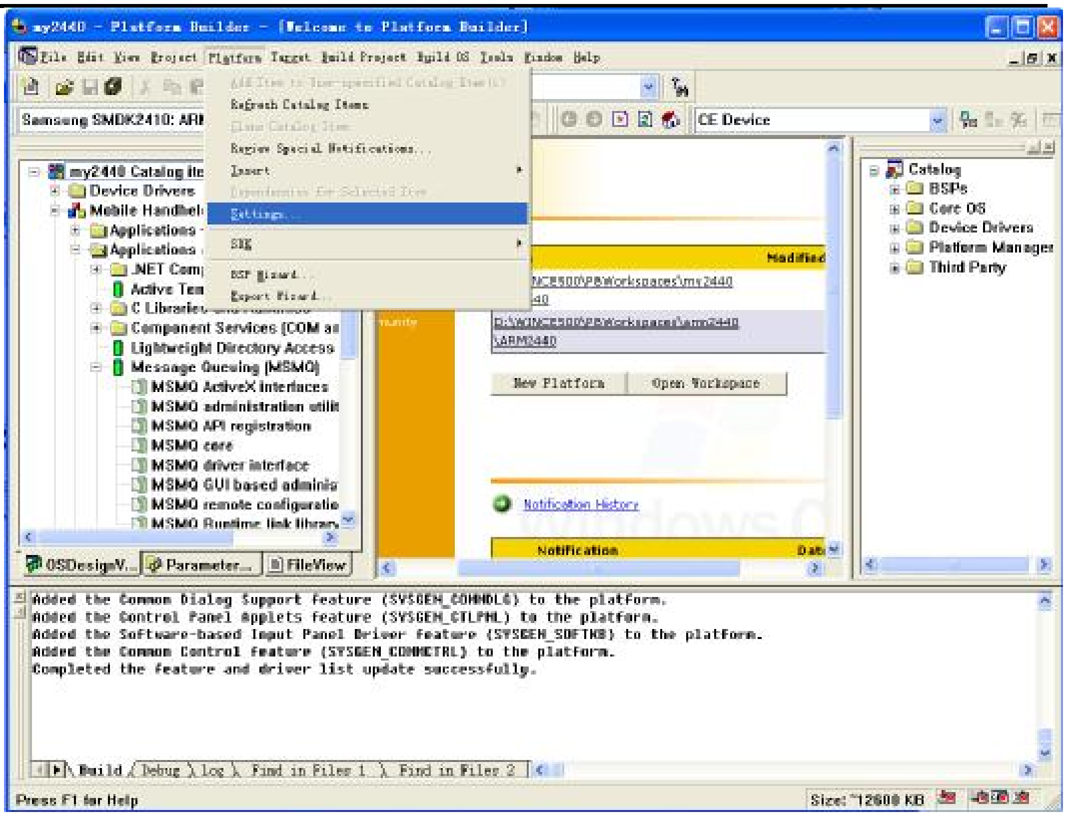
(8)出现向导结束窗口，点“Finish”结束向导:

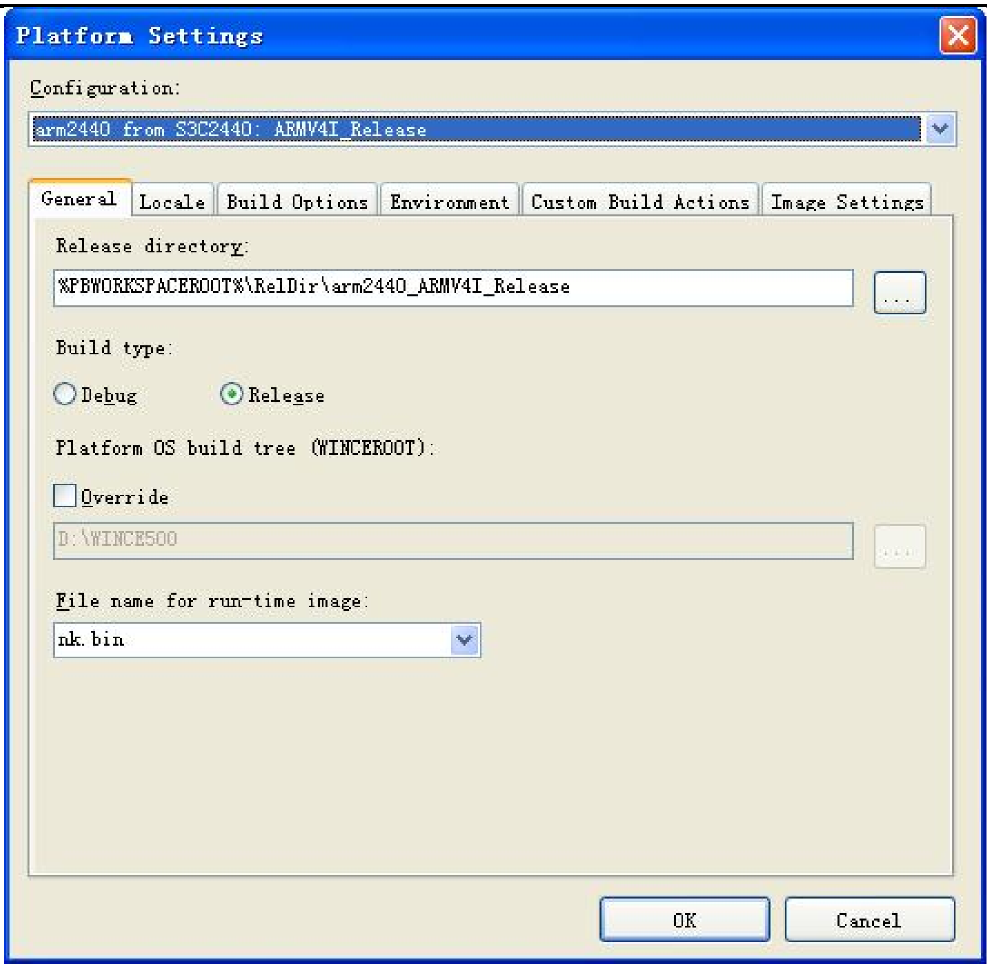


(9)现在回到PB5 的主窗口，可以看新的工程文件已经创建，我们还需要为编译做一下准备:

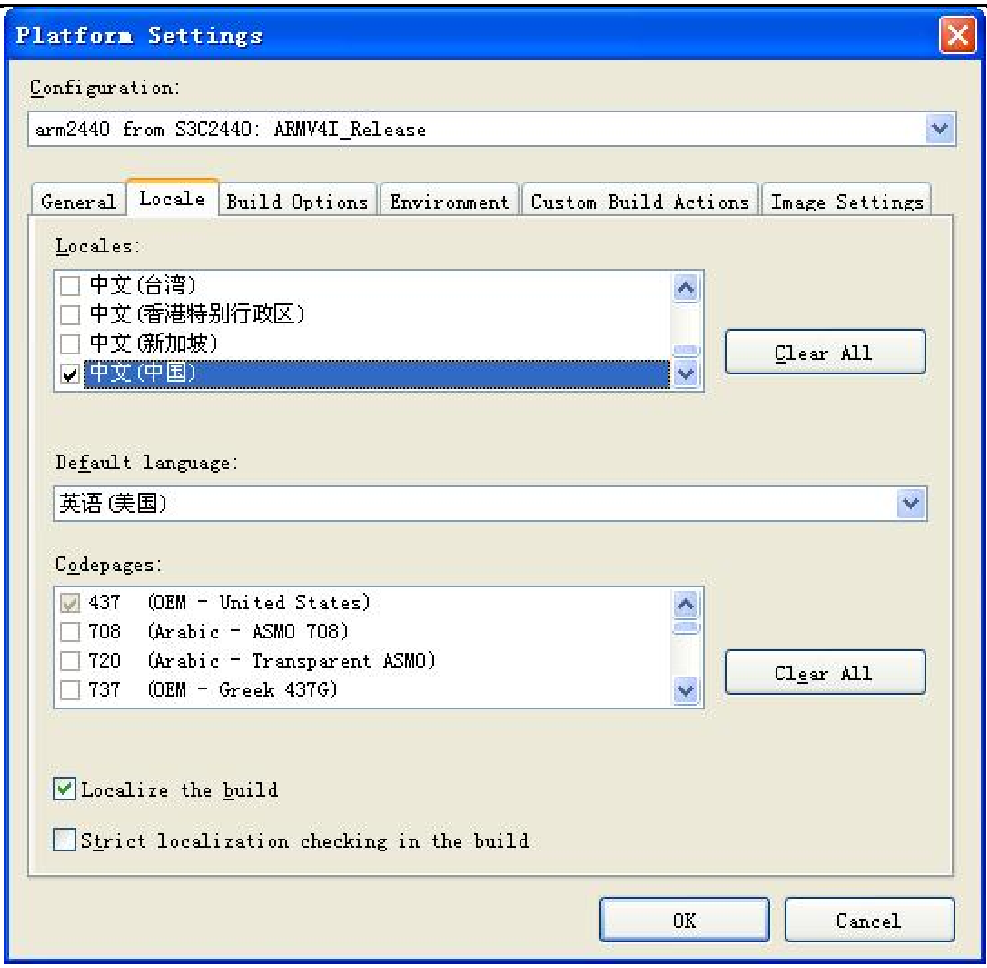


(10).点【Platform】|【Setting…】，打开工程设置窗口：

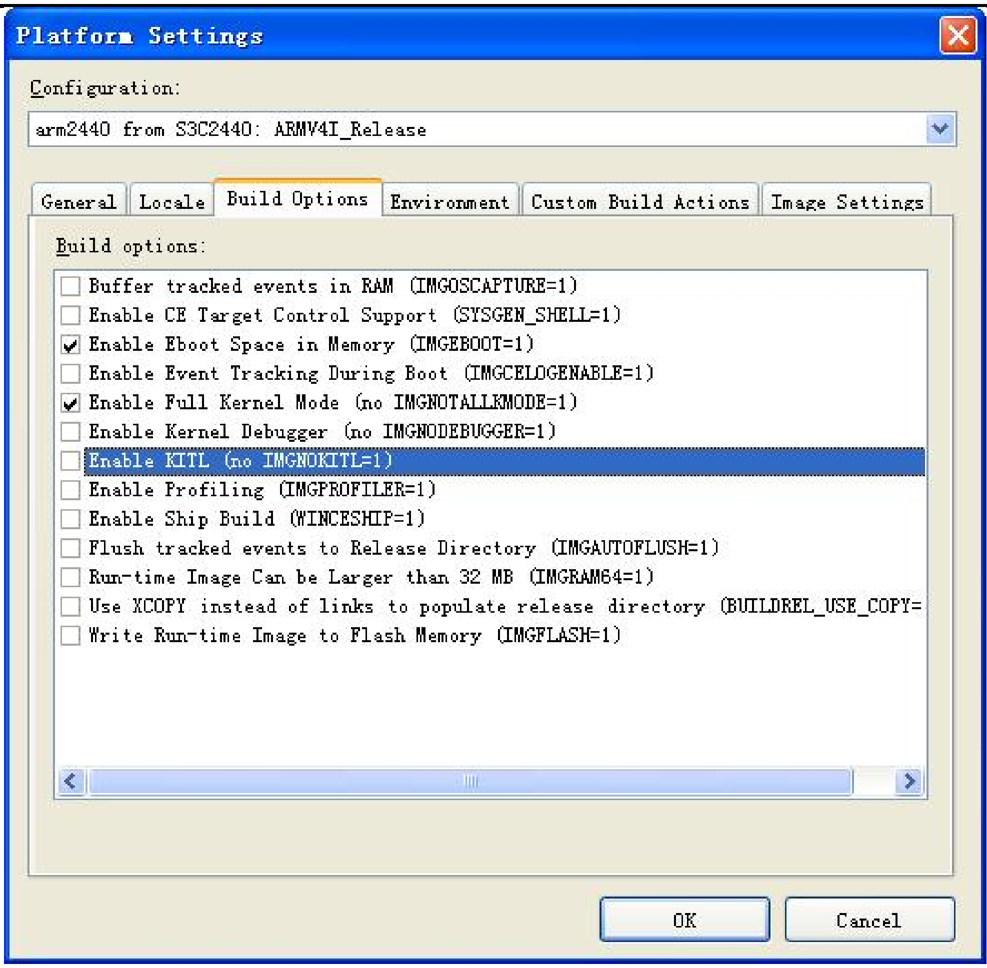

my2440 - Platform Builder - [Welcome to Platform Builder]
File Edit View Project Platform Target Build Project Build OS Tools Window Help
Refresh Catalog Items
Review Special Notifications...
Insert
Settings...
SDK
BSP Wizard...
Export Wizard...
Samsung SMDK2410: ARM
CE Device
my2440 Catalog ite
Device Drivers
Mobile Handheld
Applications
.NET Comp
Active Tem
C Libraries
Component Services (COM an
Lightweight Directory Access
Message Queuing (MSMQ)
MSMQ ActiveX interfaces
MSMQ administration utilit
MSMQ API registration
MSMQ core
MSMQ driver interface
MSMQ GUI based adminis
MSMQ remote configuratio
OSDesignV...
Parameter...
FileView
Catalog
BSPs
Core OS
Device Drivers
Platform Manager
Third Party
Modified
New Platform
Open Workspace
Notification History
Notification
Added the Common Dialog Support feature (SYSGEN_COMMDLG) to the platform.
Added the Control Panel Applets feature (SYSGEN_CTLPNL) to the platform.
Added the Software-based Input Panel Driver feature (SYSGEN_SOFTKB) to the platform.
Added the Common Control feature (SYSGEN_COMMCTRL) to the platform.
Completed the feature and driver list update successfully.
Build
Debug
Log
Find in Files 1
Find in Files 2
Press F1 for Help



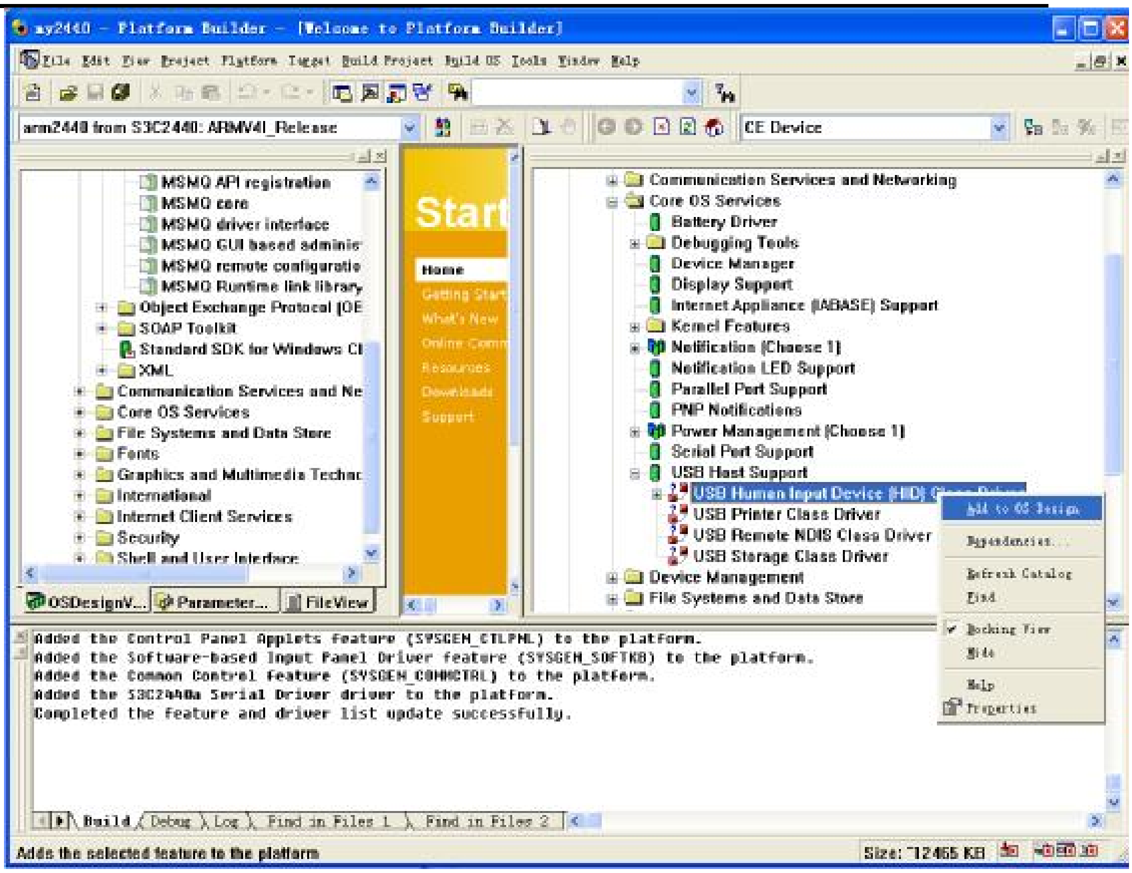
(11)在设置窗口中，点“Locale”选项卡来设置目标内核的语言，这里选择中文:



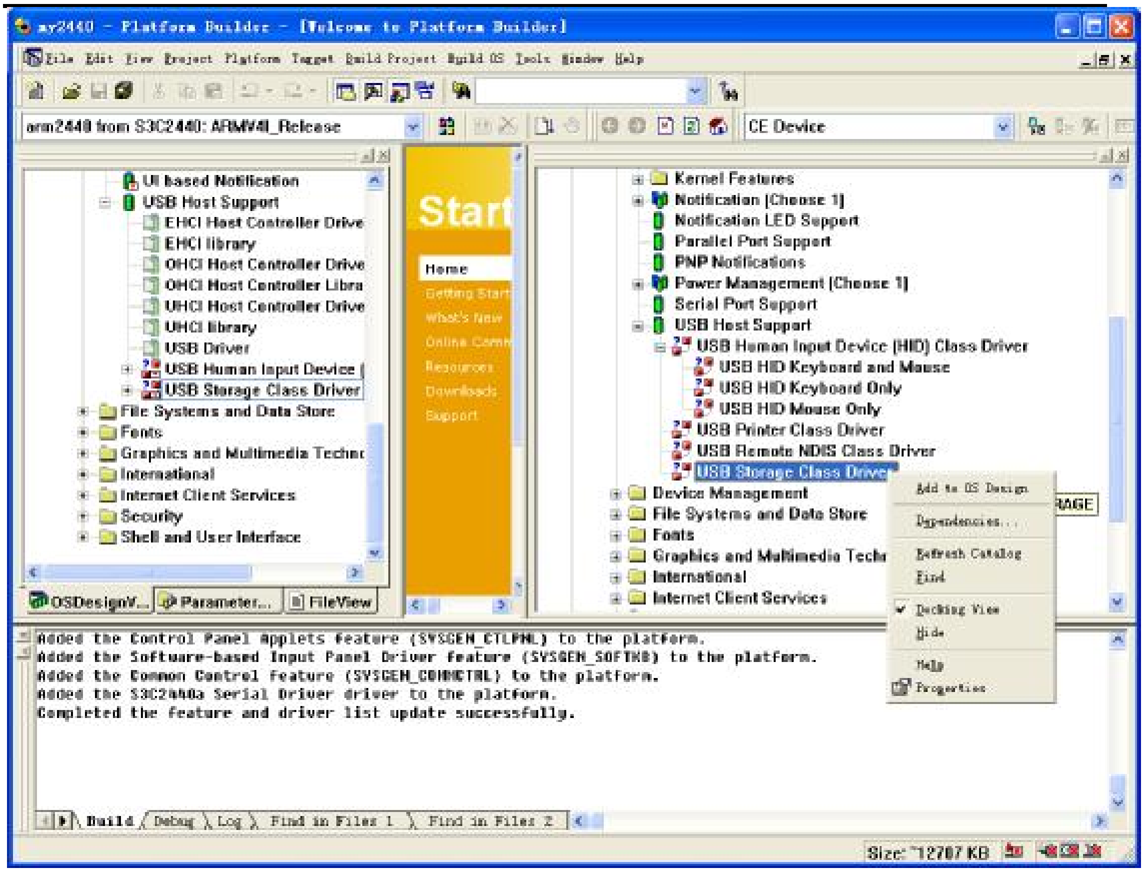
(12).点“Build Options”选项卡，去掉“Enable CE Target Control Support”和“Enable KITL”这两个设置，其他使用缺省，并点“OK”完成设置，如图：



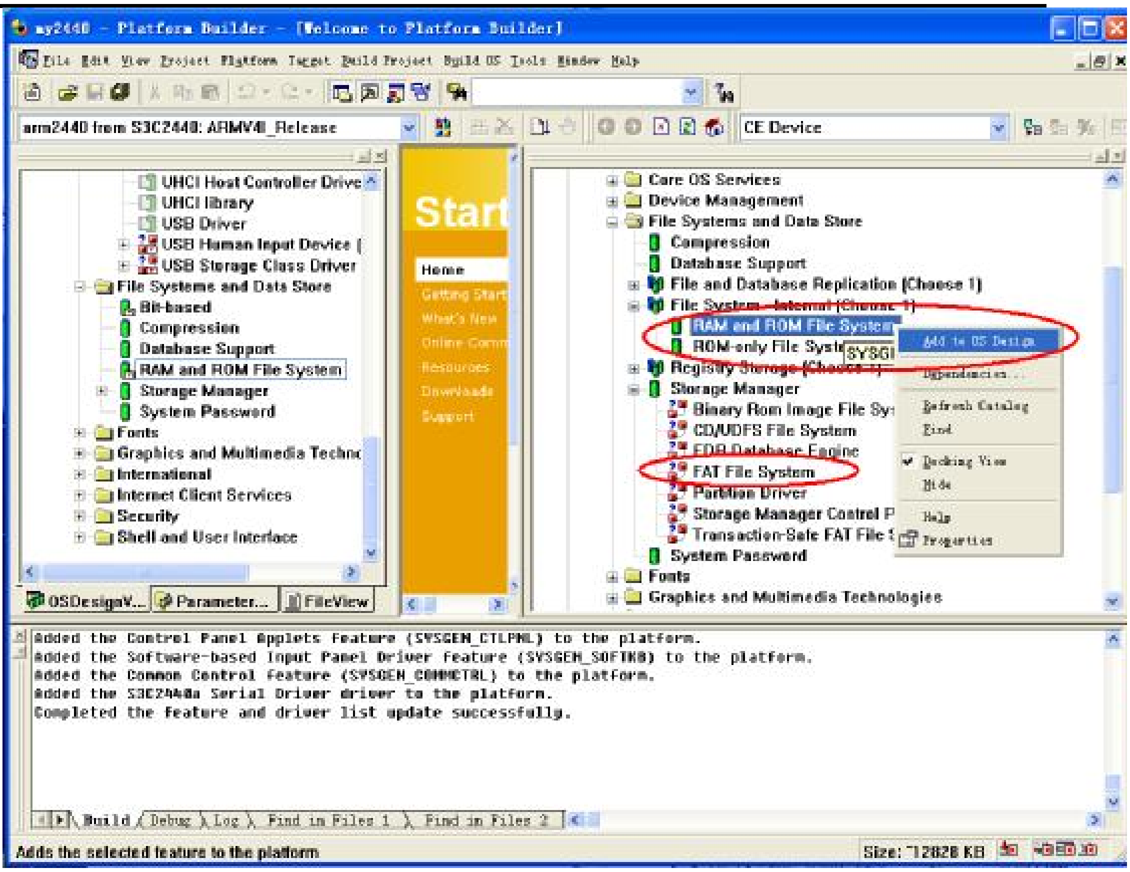
(13)加入USB 鼠标和键盘的支持，在Catalog 一栏依次点击展开【Core OS】|【Windows CE device】|Core OS Services】|【USB Host Support】|【USB Human Input Device(HID) Class Driver】，点右键选择“Add to OS Design”，并展开其子项添加“USB HID Keyboard and Mouse”，如图：



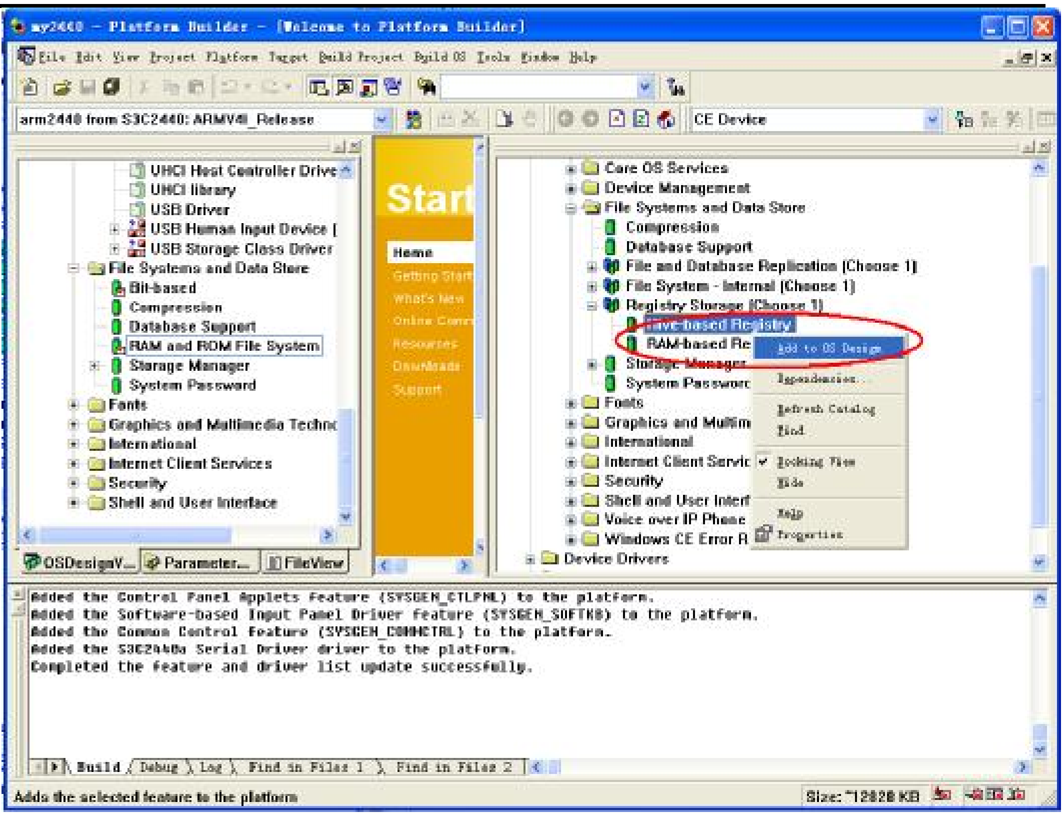
(14)使用同样的方法可以添加USB 存储设备的支持，如图:



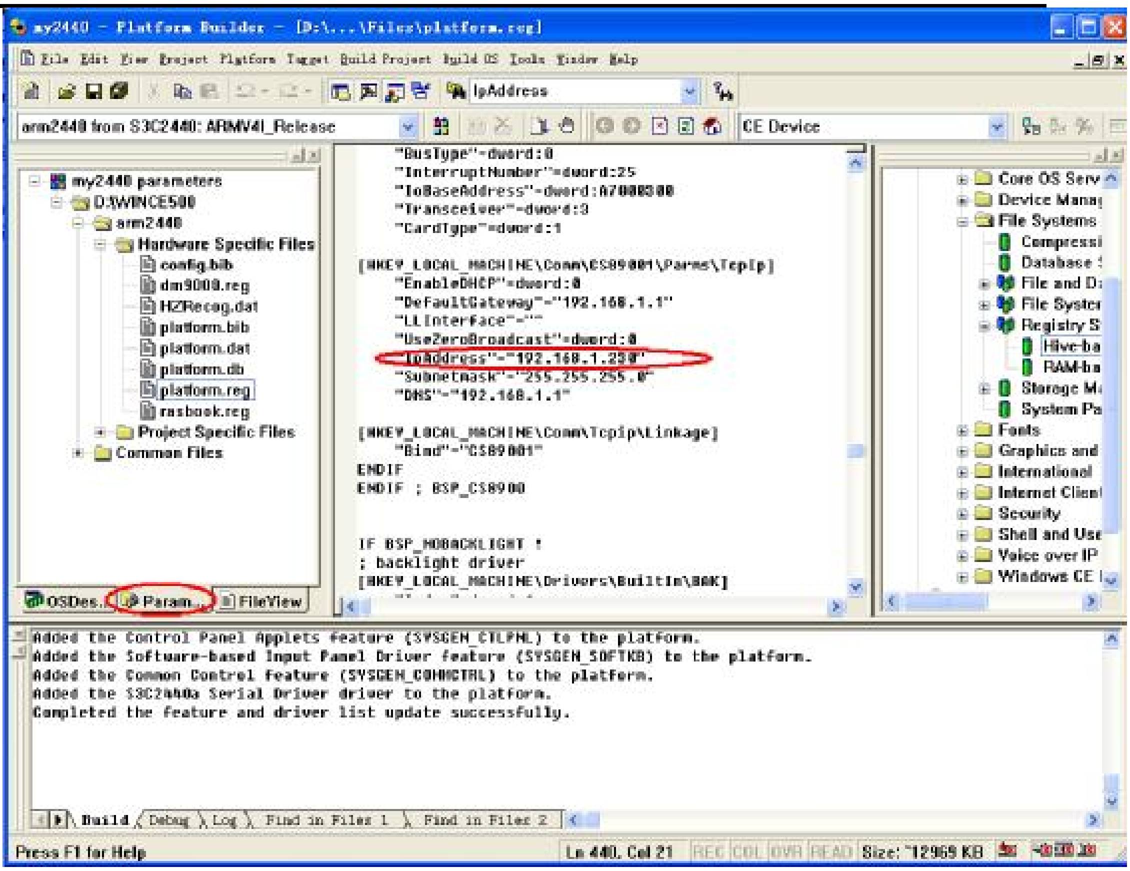
(15)添加文件系统的支持，点【Core OS】|【File Systems and Data Store】|【File System–Internal (Choose I)】|【RAM and ROM File System】，为了支持优盘所使用的 FAT32 文件系统，你还需要添加图中矩形方框中的**FAT File System**，如图：



(16)添加注册表保存卡支持，点【Core OS】|【File Systems and Data Store】|【Registry Storage (Choose I)】|【Hive-based Registry】，如图：



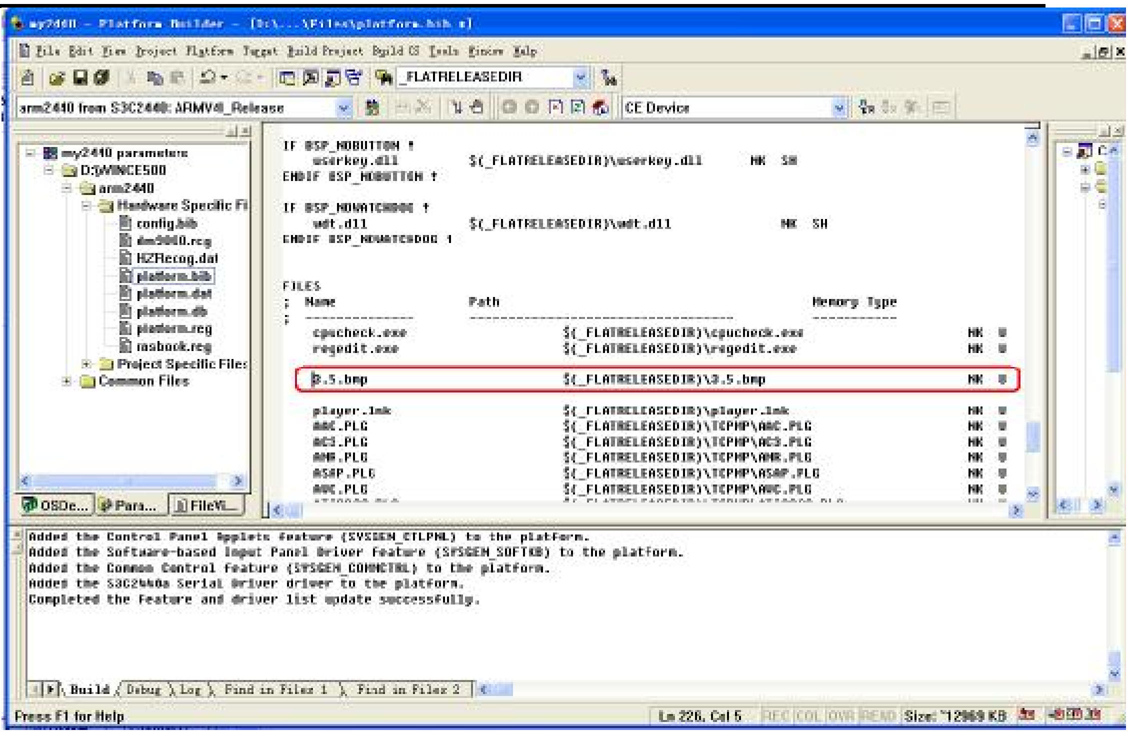
(17)修改默认的IP 地址，打开platform.reg 文件，找到如图红色框所示的位置，在这里你可以修改开机后默认的IP 地址，网关设置，DNS 等设置：



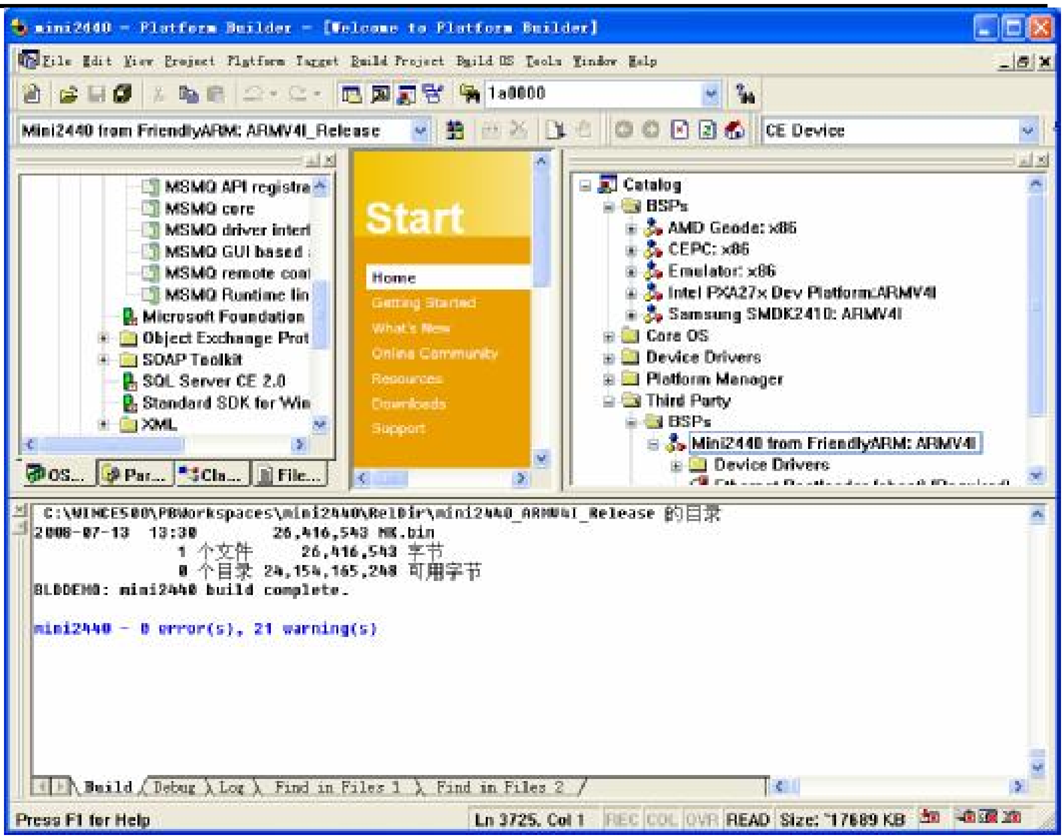
(18)修改桌面背景图片， 准备一个bmp格式的图片， 复制到D:\WINCE500\PLATFORM\GQ2440\Files 目录中，并命名为3.5.bmp，如图：



打开platform.bib 文件，在FILES 栏目中把bliss.bmp 添加进去，如图:



(19)保存以上设置，点【File】|【Save】即可，再点【Build OS】|【Sysgen】或者点工具栏的按钮就开始编译内核了，编译完毕如图所示：



关于更多的内核配置和更改用户可以自己摸索尝试，或者参考相关网上资料等，在此不再叙述。

9.1.9 BootLoader 之 Nboot 的编译和烧写

说明：我们提供的 Nboot 可同时适用于WindowsCE5/6，因此在光盘中的所有Nboot 项目源代码都是相同的，只不过为了方便组织使用，特意增加了拷贝，并分别放在了不同的目录。

编译 Nboot 需要使用ADS 集成开发环境，详细的步骤见本手册第四章，在Windows 7系统上，你可以通过安装Windows XP Mode 来创建Windows XP 环境。

Nboot 是一个十分简单的bootloader，其大小不到4K，一般被烧写到Nand Flash 的Block 0 位置用来启动WinCE 内核，Nboot 原由三星提供，我们对此做了很多改进，目前有如下特色功能:

- 自适应支持 64M/128M GQ2440
- 支持开机画面快速显示
- 支持加载 WinCE 内核的动态进度条
- 启动 WinCE 仅需5-10 秒，视内核大小而定

需要注意的是，Nboot 并不具备烧写功能，它只能读取已经烧写处理好的文件：开机画面(BootLogo)和WinCE 内核。

Nboot 具有很方便的定制性，你可以通过头文件定义修改开机画面的显示位置、背景，以及进度条的颜色、位置、长宽等，这些定义位于option.h 文件中，如下:

```
//通过更改定义，选择相应的LCD型号，此处默认选择S35，表示三星3.5"LCD
#define LCD_S35
//#define LCD_a43
//#define LCD_a56
//#define LCD_A70
//#define LCD_VGA1024768

//设置背景色
#define BACKGROUND_R    0x00
#define BACKGROUND_G    0x00
#define BACKGROUND_B    0x00

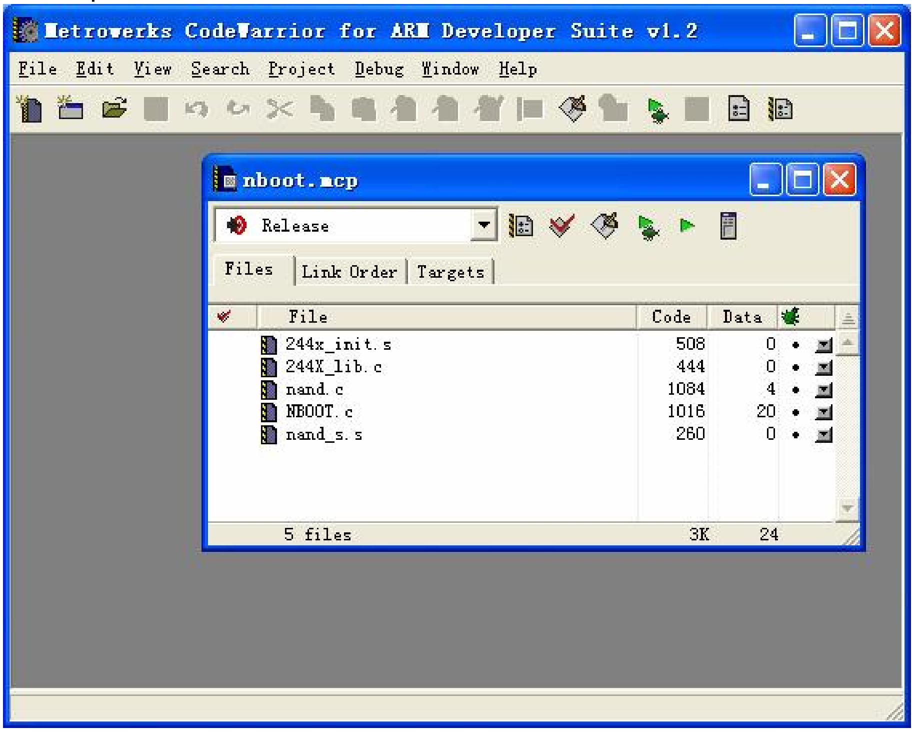
//设置进度条的颜色
#define PROGRESS_BAR_R 0xFF
#define PROGRESS_BAR_G 0xFF
#define PROGRESS_BAR_B 0x00

//设置开机图片的位置
#define LOGO_POS_TOP      0
#define LOGO_POS_LEFT     0
```

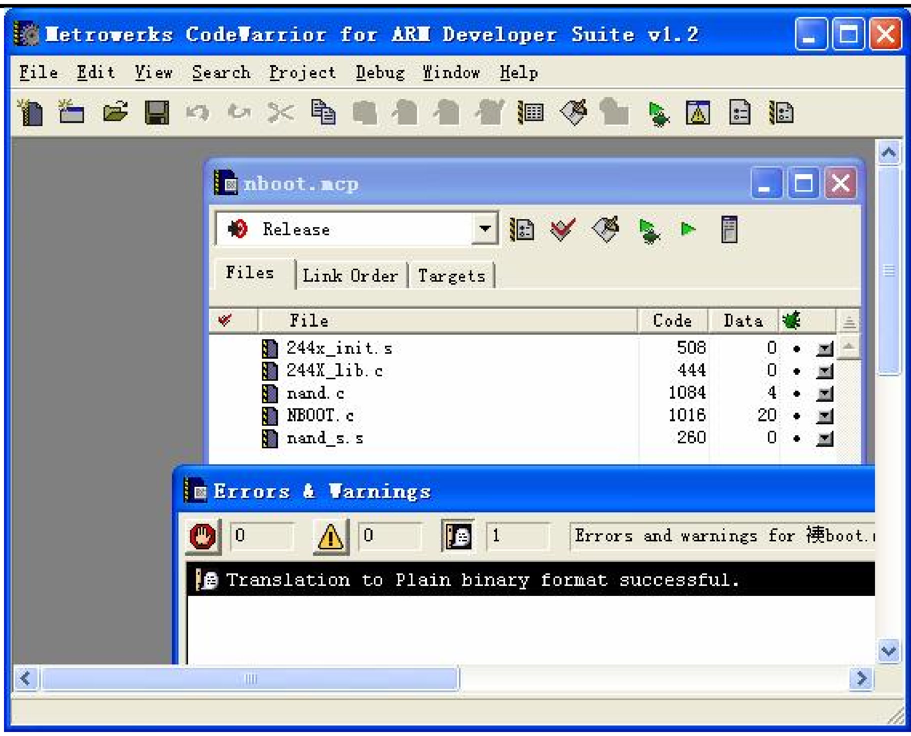
```
//设置启动条的位置和长宽
#define PROGRESS_BAR_TOP      234
#define PROGRESS_BAR_LEFT      5
#define PROGRESS_BAR_WIDTH    310
#define PROGRESS_BAR_HEIGHT    6
```

编译 Nboot

把光盘中“WindowsCE5.0”目录中的文件夹“nboot”文件夹复制到硬盘的某一个目录(在此为D:\work)，去掉其只读属性，运行ADS1.2 集成开发环境，点【File】|【Open…】打开nboot.mcp 文件，如图。



这时点菜单【Project】|【Make】或者直接按F7 键，开始编译nboot 项目，编译完毕如图：



在 **D:\work\NBOOT\nboot_Data\DebugRel** 目录下会生成**nboot.bin** 可执行文件，如图。

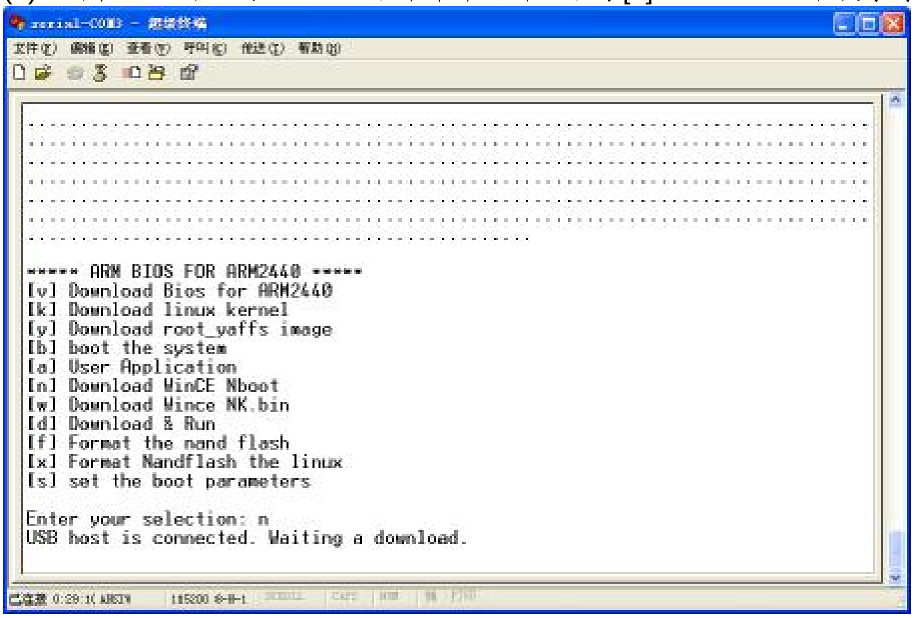
9.1.10 把 NBOOT 烧写到 Nand Flash

(1)连接好开发板电源，串口线，USB线，并**通过Nor_KEY按钮使开发板从Nor Flash 启动**，分别打开串口超级终端和DNW，上电启动开发板。

(2)保证USB 驱动已经安装好(前面已经详细介绍了USB 驱动的安装方法)，这时可以看到DNW 的标题栏显示[USB: OK]，如果没有安装好驱动会显示[USB: x]，如图所示：



(3)这时在超级终端的BIOS 功能菜单中选择功能号[n]，出现USB 下载等待提示信息:



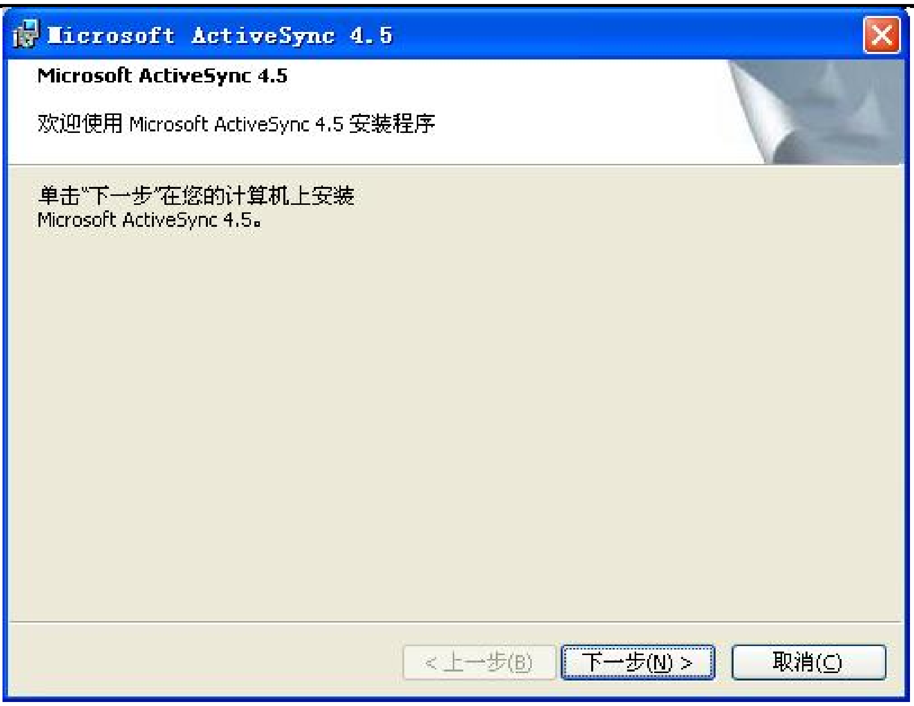
(4).点击DNW 程序的“USB Port”□“Transmit”，如图选择刚刚编译出的映象文件(光盘“images\wince5.0”目录中有已经编译好的可执行文件)，这样就开始下载了，瞬间就可下载完毕，armvivi 会把它自动烧写到Nand Flash 起始block 0 中；通过POW_KEY按钮使开发板从Nand Flash启动系统；如果系统中已经烧写好了WINCE 内核映象文件，马上就会开机画面和进度条，稍等片刻就看到wince 启动了。

9.2 使用 ActiveSync 与 PC 同步

使用微软提供的工具 ActiveSync，可以让开发板与 PC 之间十分方便的进行通讯连接，从而实现文件上传，远程调试等功能。

9.2.1 安装 ActiveSync

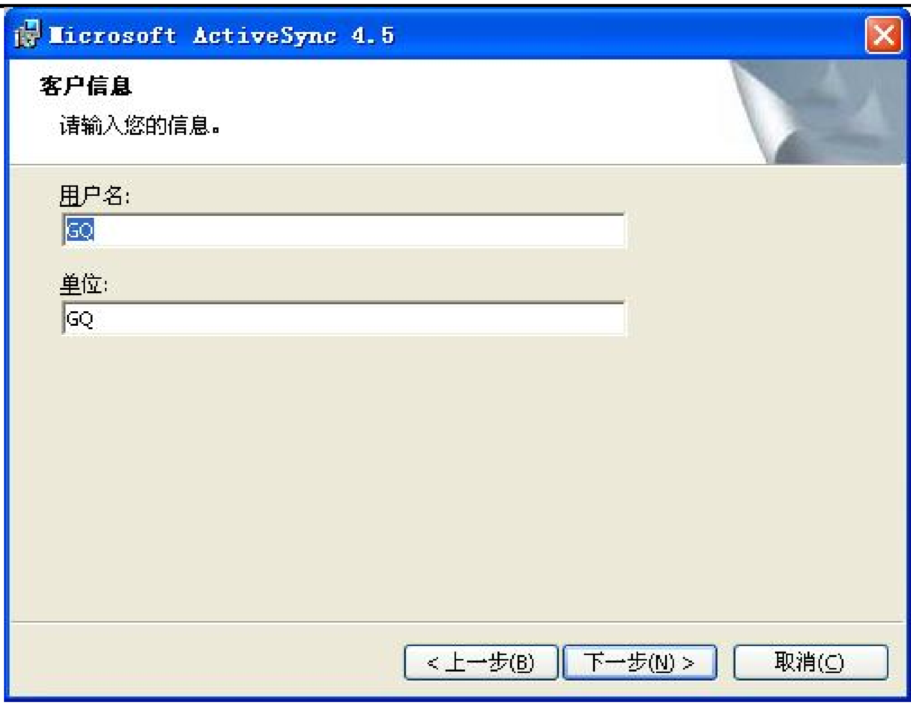
在光盘的“windows 平台工具”目录中**\ActiveSync_4.5**文件夹，双击运行里面的MicrosoftActiveSync_setup_cn.msi 开始安装。



如图选择"我接受该许可协议中的条款"，点"下一步"继续



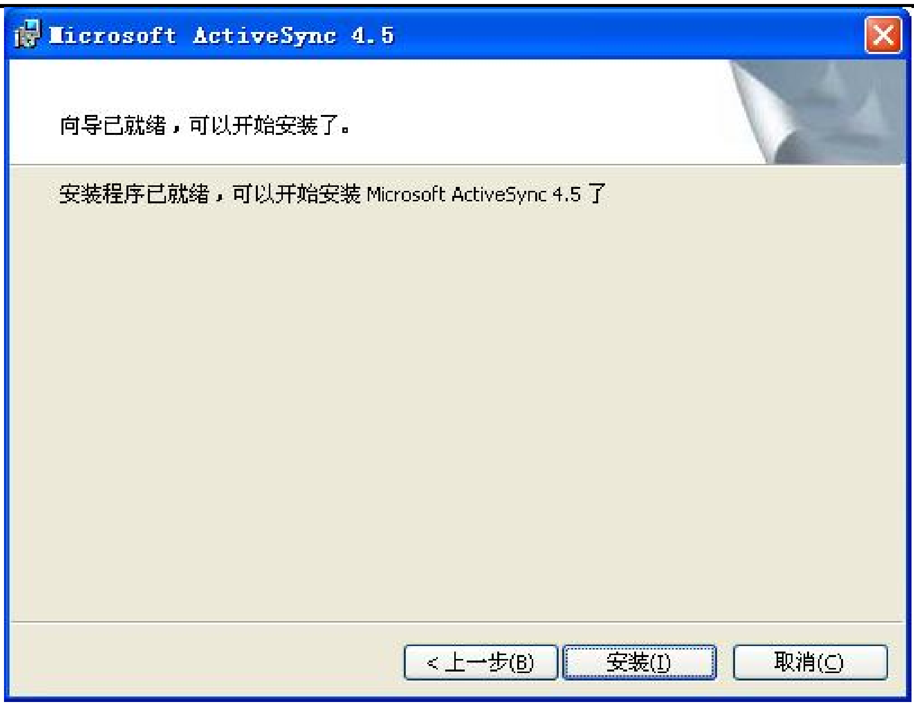
输入用户名和单位名称，点“下一步”继续



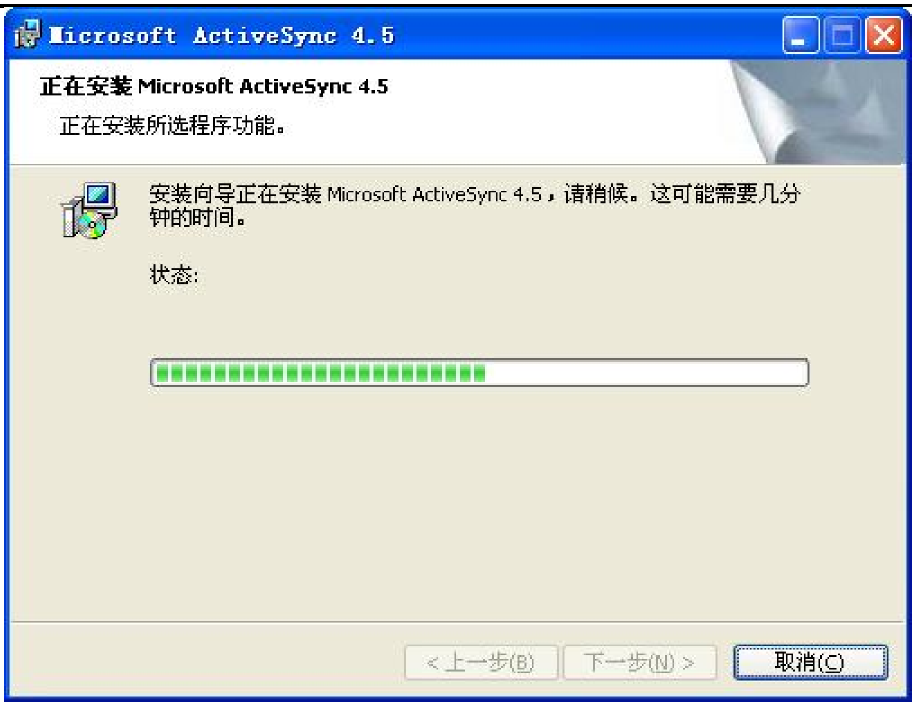
选择要安装的目的路径，点“下一步”继续。



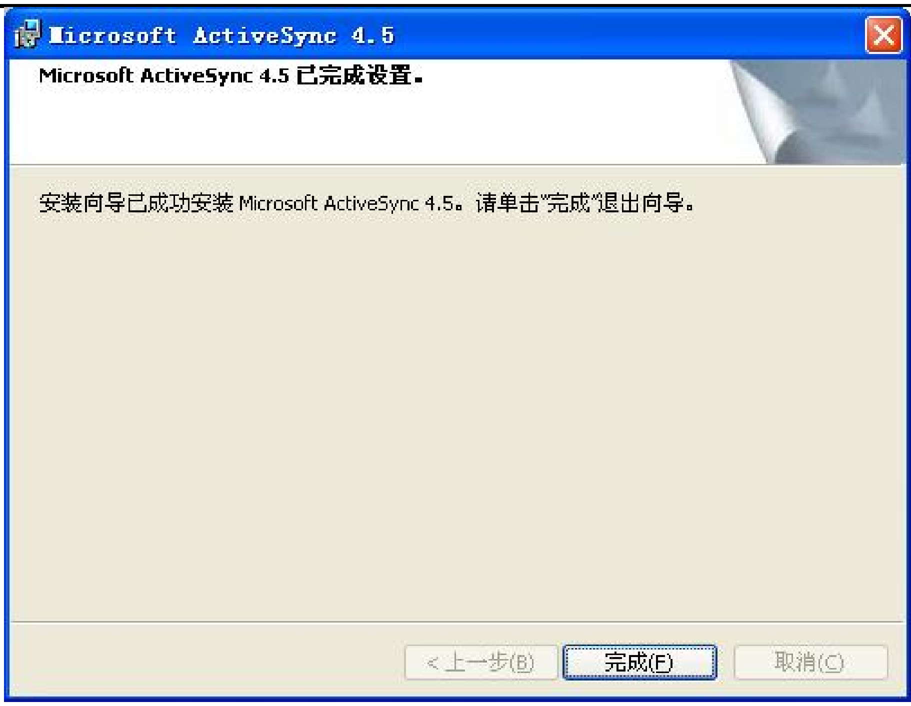
出现如下界面，点"安装"开始进行安装。


Microsoft ActiveSync 4.5
向导已就绪，可以开始安装了。
安装程序已就绪，可以开始安装 Microsoft ActiveSync 4.5 了
< 上一步(B)
安装(I)
取消(C)

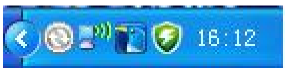
出现安装过程界面，如下



安装完毕，点“完成”安装完毕。



这时会在任务栏出现相应的图标托盘，出现如下界面



9.2.2 为同步通讯安装 USB 驱动

确认板子里面已经烧写好了我们提供的 WINCE 映象文件，并开机运行，系统起来以后，接上USB 电缆，并与PC 连接，如果以前没有安装过这个驱动，计算机会出现"发现新硬件"的提示，这时就可以按照本小节的步骤安装驱动了，我们用到的驱动程序的位置在光盘的**\Windows 平台工具\CE 用同步USB 驱动**目录中：

Step1: 找到新的硬件，请如图进行选择，点"下一步"继续。



Step2: 选择“从列表或指定位置安装”，点“下一步”继续，如图。



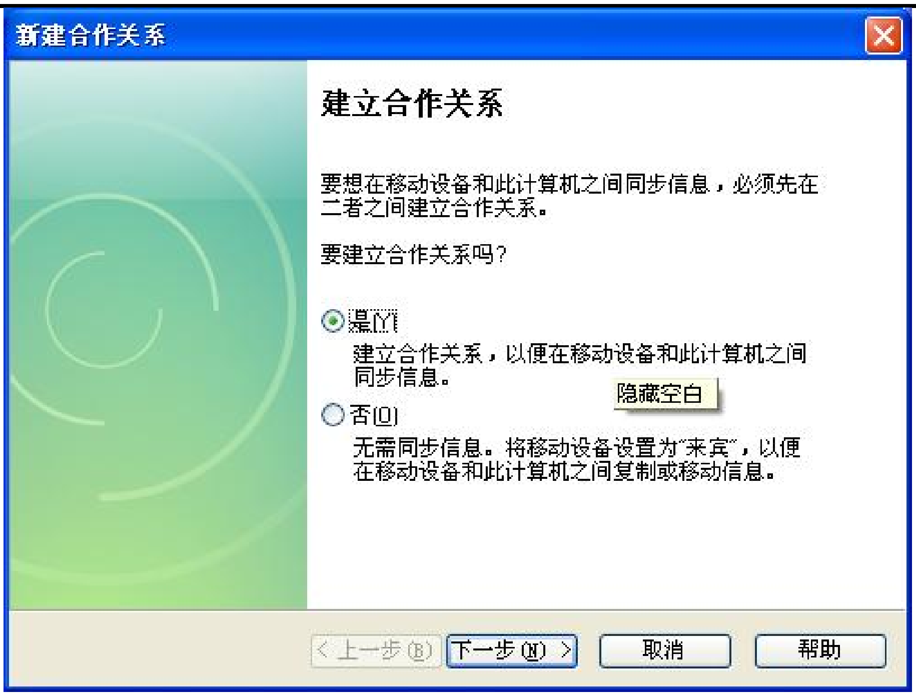
Step3: 点“浏览”并定位到光盘**\Windows** 平台工具**\CE** 用同步**USB** 驱动，如图



Step4: 系统将会自动搜索一会，然后出现如下界面，点"仍然继续"，系统将自动安装完毕USB 驱动。



Step5: 同时ActiveSync 会自动跳出运行，这时，您可以按照设置向导按默认选项进行设置。如果您对使用ActiveSync 还不熟悉，请点击“取消”。



在完成同步设置后会出现如下界面，已完成连接与同步：

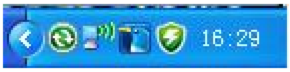


9.2.3 使用 ActiveSync 同步传输工具复制文件

经过上一小节的 USB 驱动安装，实际上您已经看到开发板已经同PC 连接好了，我们现在看看如何使用它。

(1)查看PC 端的连接情况

当我们连接好开发板与PC后，我们也可以注意到PC 任务栏的右下角的ActiveSync 也变成了绿色，这说明一切准备就绪。



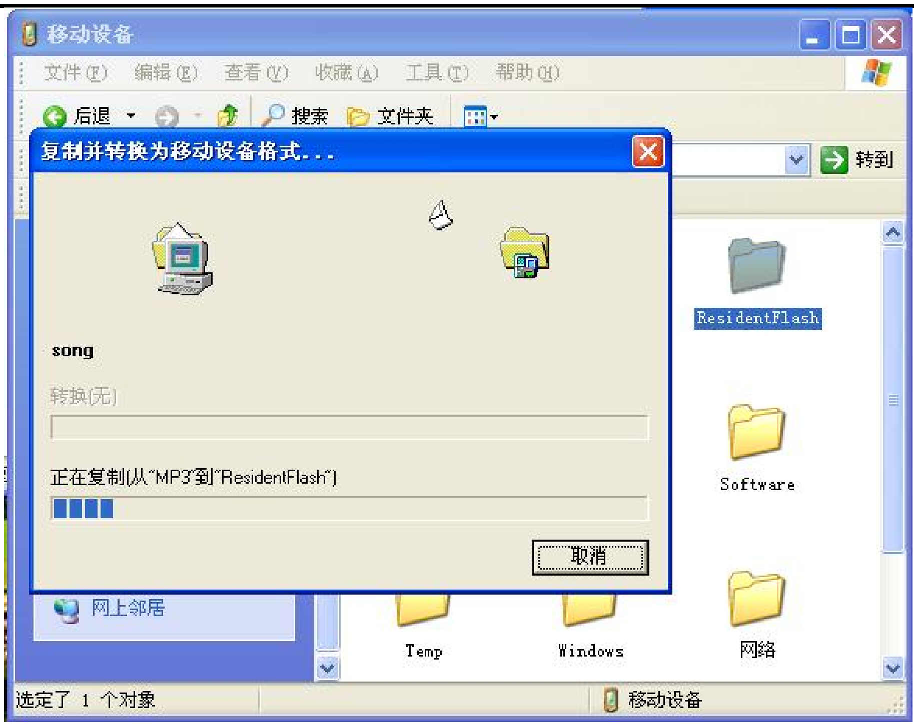
实际上，ActiveSync 安装完毕后，在“我的电脑”里会出现一个“移动设备”图标，现在我们双击打开它，您将看到目标板的所有目录，如下


我的电脑
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) 我的电脑 转到
链接
系统任务
查看系统信息
添加/删除程序
更改一个设置
其它位置
网上邻居
我的文档
共享文档
控制面板
详细信息
WIN7 (C:)
SOFT (D:)
DATA (E:)
BKUP (F:)
DVD 驱动器 (I:)
WINXP (G:)
共享文档
DVD/CD-RW 驱动器 (H:)
Administrator 的文档
移动设备
选定 1 个对象
我的电脑

点击"移动设备"进入:



这时，双击打开 ResidentFlash 目录，在您的电脑里面找一首mp3 文件或者其他小于30M 的文件并拖动到打开的ResidentFlash 文件夹，如图



这样，我们就能像操作 Windows 里面的目录一样，对开发板里面的目录进行文件复制等操作了。

(2)查看开发板一端的情况

接上一步，我们复制了一首 Mp3 文件到ResidentFlash 目录，是不是真的有了呢？

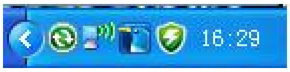
接上 USB 鼠标或者使用触摸笔，点击打开【我的设备】|【ResidentFlash】 目录，可以看到如下界面。



9.2.4 使用 ActiveSync 与 Platform Builder 连接实现通讯与截屏

首先确认开发板已经和 PC 之间可以使用ActiveSync 连接成功，同时网络连接也没有问题(至关重要!)，如下图





我们先设置开发板上的 WINCE 连接。

注意：为了更好的设置请先按照2.6.11的方法，使屏幕旋转。

打开【开始】|【设置】|【网络和拨号连接】，点“新建连接”设置对话框中，选择连接类型为“直接连接”，如下图



改连接名称为“我的usb连接”或者不改也可以，点“下一步”，在出现的“选择设备”下拉列表中选择“USB Cable：”，如下图



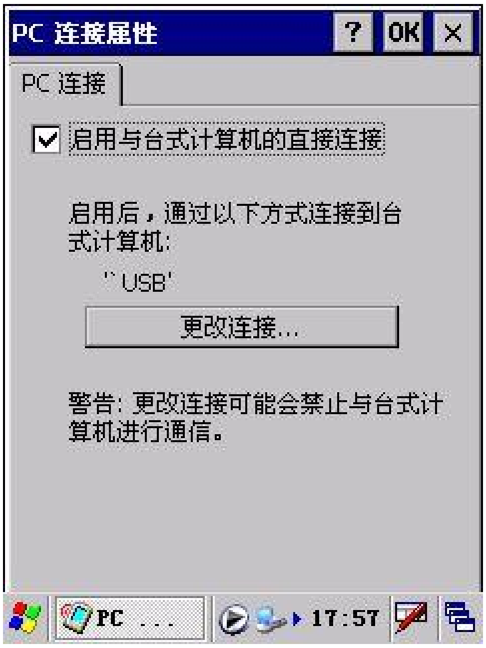


点右上角的“OK”，这时出现“我的usb 连接”图标，如下图

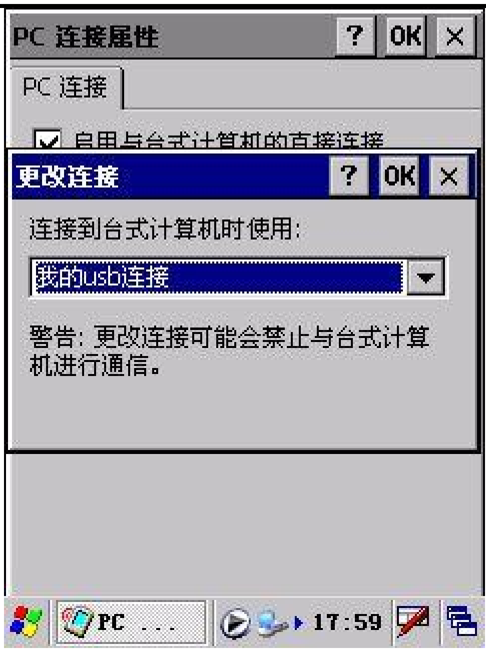


打开【开始】|【设置】|【控制面板】，双击打开“PC 连接”图标，进入“PC 连接属性”设置对话框，选中“启用与桌面计算机的直接连接”的复选框，然后再点“更改连接”按钮，如下图





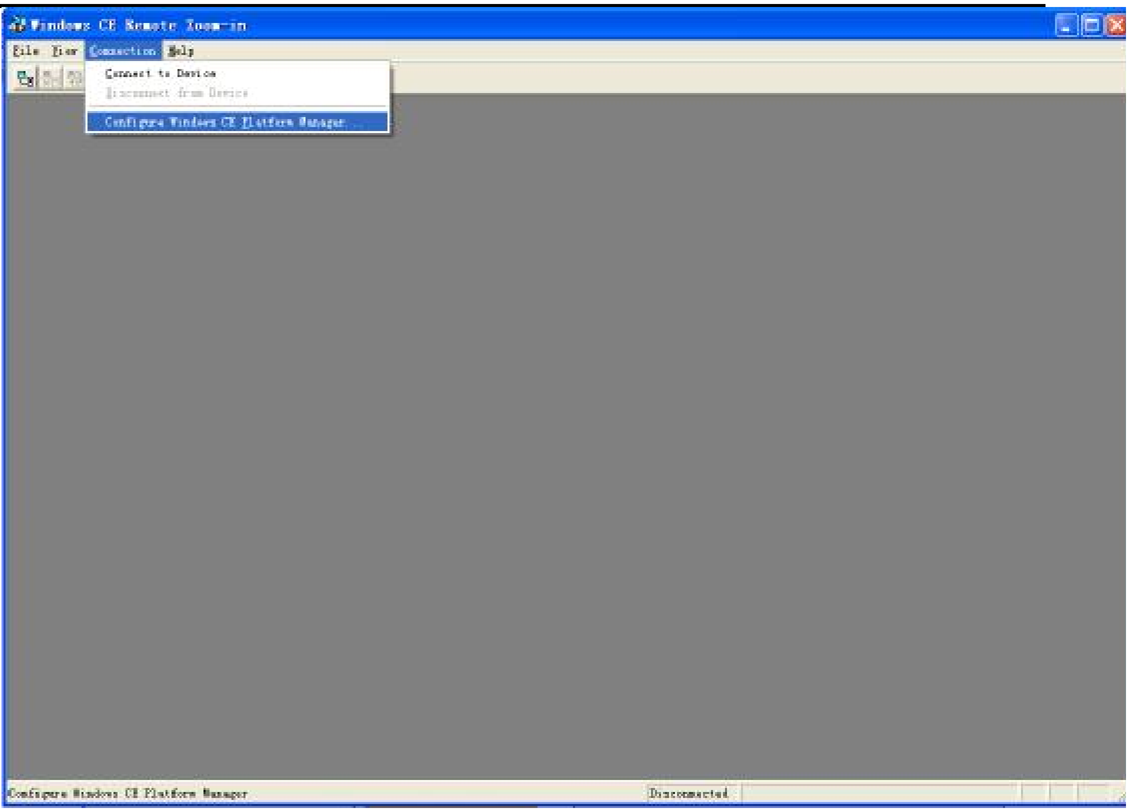
在“更改连接”设置对话框的下拉列表中选择刚刚新建的连接“我的 usb 连接”，然后点“OK”退出设置，这样就完成了开发板这端的设置。



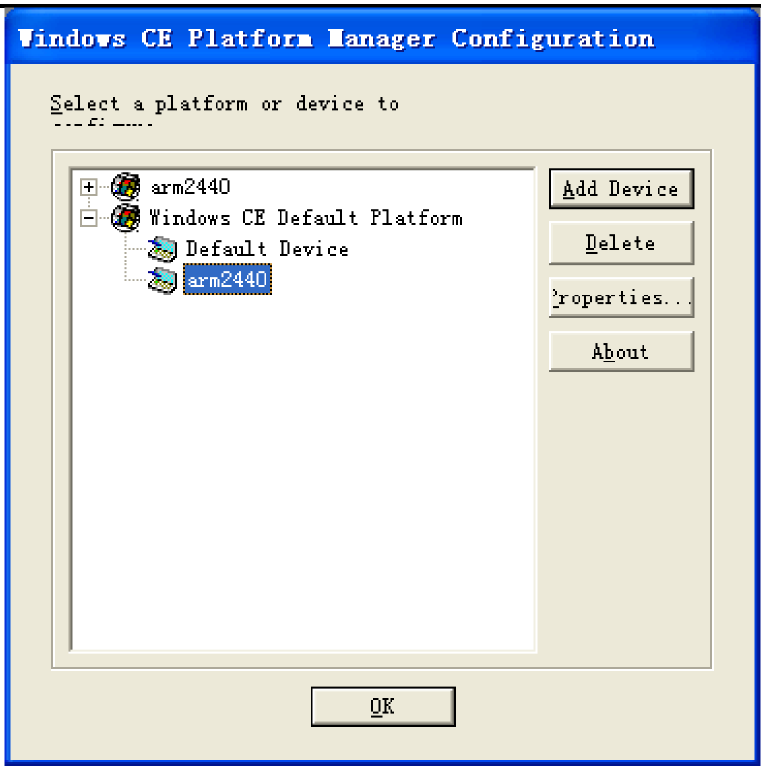
现在，我们再来设置 PC 端，请务必确认ActiveSync 已经和开发板连接成功。

点 PB 菜单【Tools】|【Remote zoom-in】，打开远程图片缩放工具，这个程序可以实现对远程移动设备显示屏幕的截屏。

Remote zoom-in 窗口打开之后， 先要配置一下平台管理器，点击菜单【Connection】|【Configure windows ce platform manager】，如下图

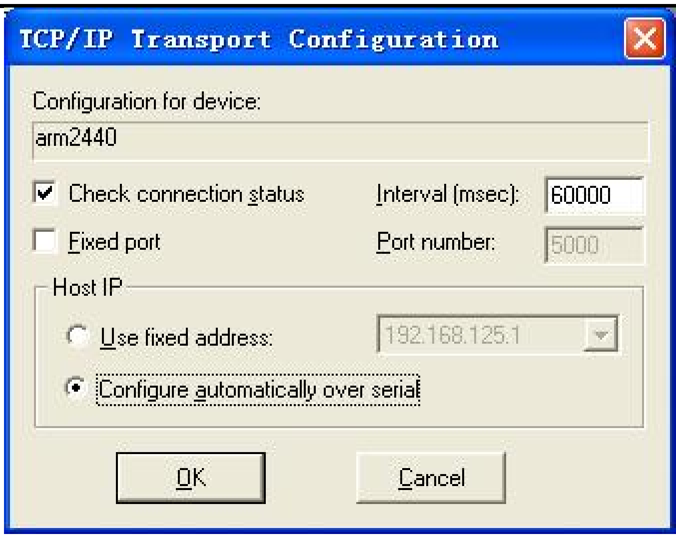


在弹出的对话框中添加一个新设备，并且取名为“GQ2440”，如下图



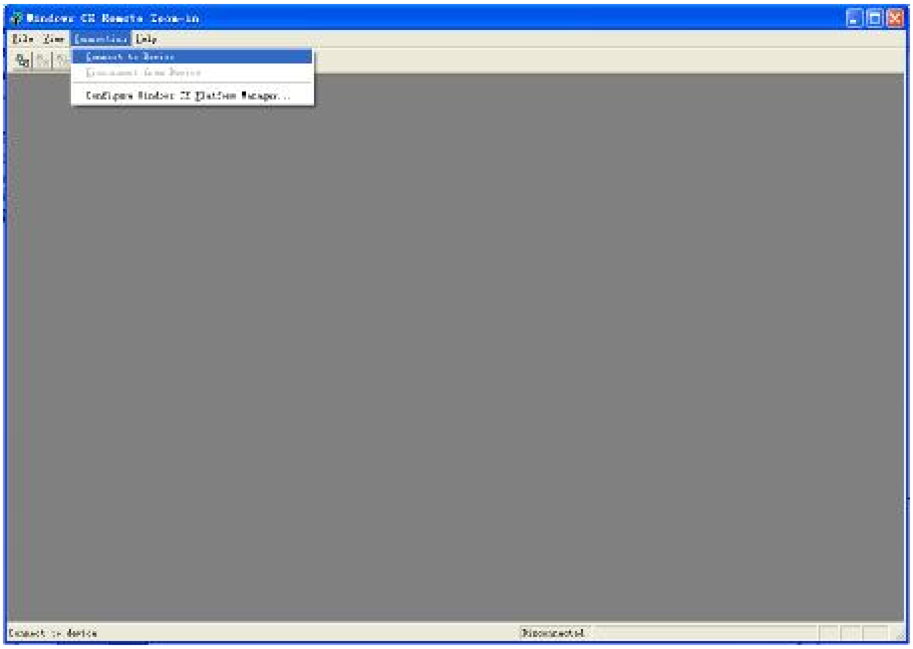
点右边的“Propertions…”按钮，设置“GQ2440”设备平台的属性，如下图

点“Transport”下拉框右边的“Configure”按钮，开始设置TCP/IP 传输，在HOST IP一项中输入您的主机IP 地址，请保证开发板WINCE 的IP 地址(默认为192.168.1.230)与PC主机的IP 地址在同一个网段，也可以自动匹配，如图



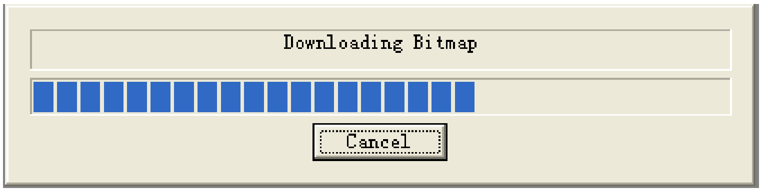
设置结束，点OK 按钮返回到Remote Zoom-in 主窗口，点菜单Connection->Connect to

Device，开始连接开发板。

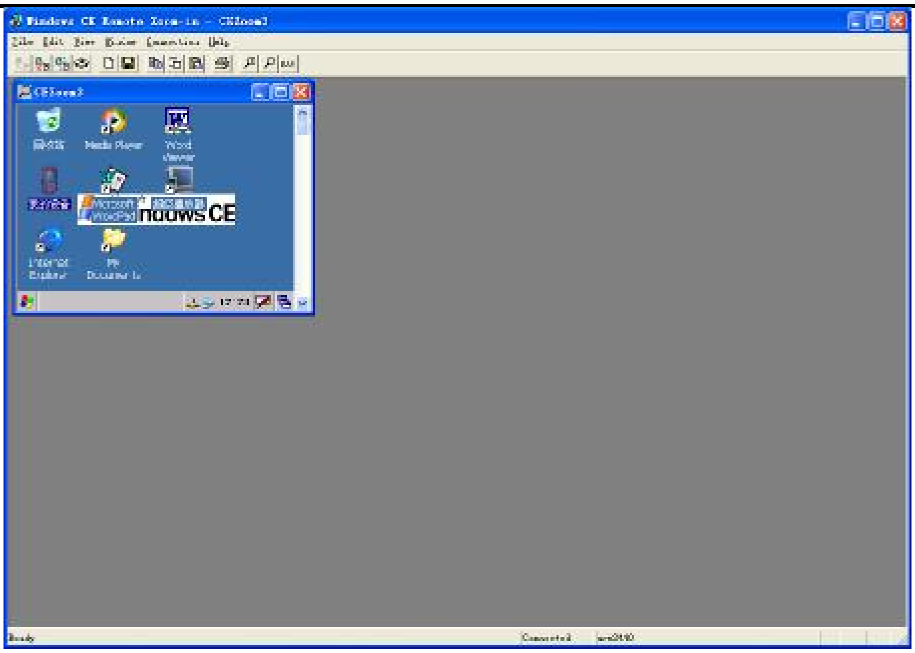


这时一般会出现如下图所示界面，显示正在连接。





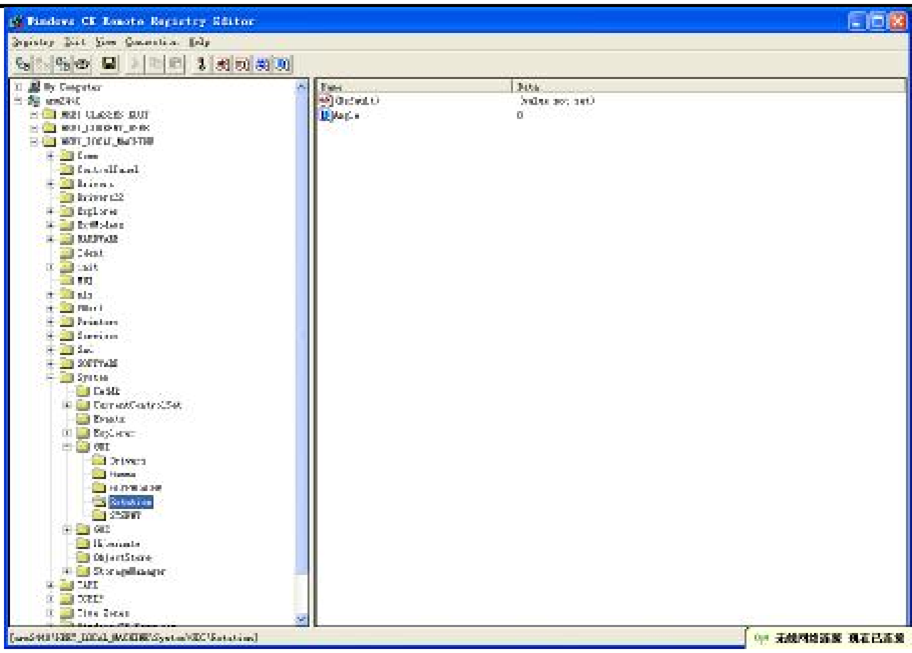
当连接成功，会出现和目标板当前屏幕一样的几个界面窗口，如下图



这时点 File-Save as 可以保存当然的屏幕截图。

9.2.5 使用 ActiveSync 与 Platform Builder 在线编辑注册表

当您学会使用上一小节的内容实现截屏之后，您还可以在 PB 的Tools 菜单中点击“Remote Registry Editor”来运行远程注册表编辑工具查看并修改wince 的注册表内容，如下图，在此就不作更详细的解释了。

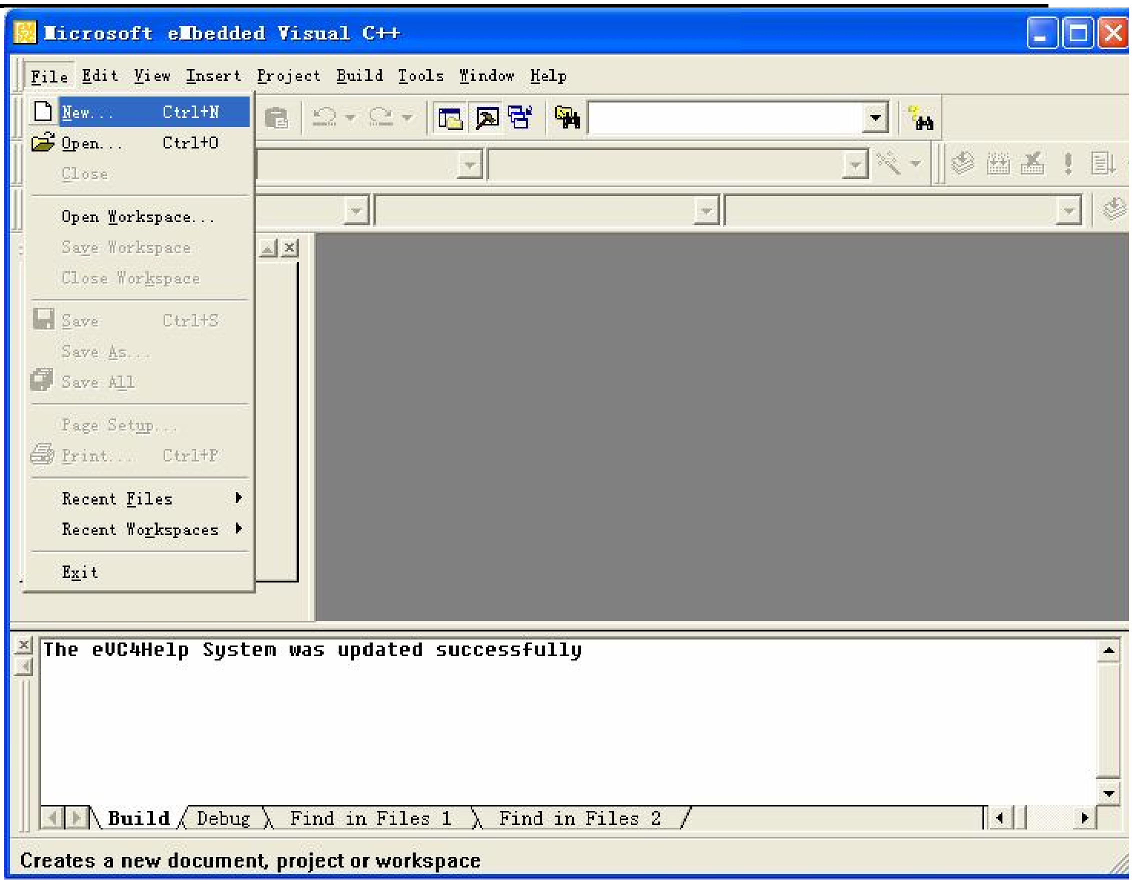


9.3 创建 EVC 的 Hello,World，并编译下载到开发板运行

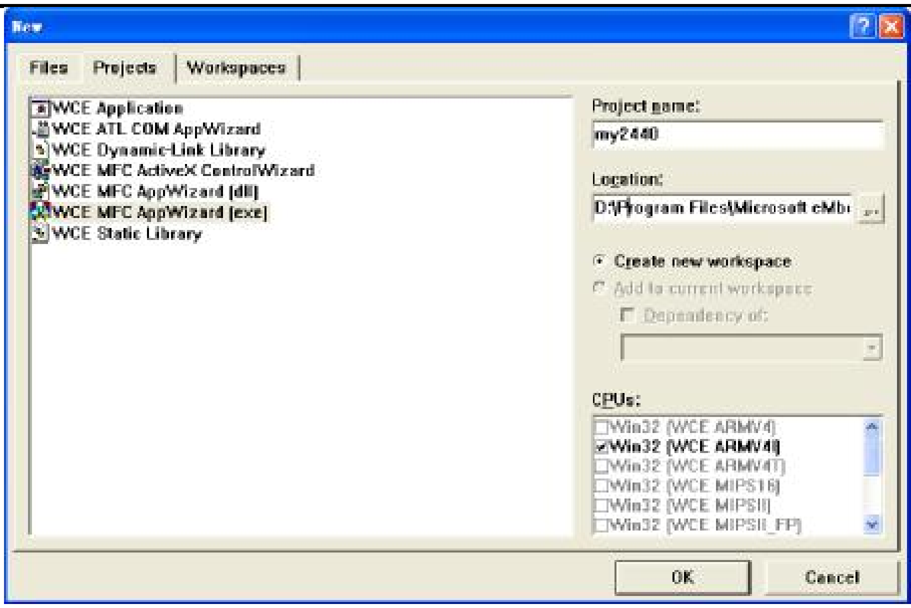
我们将不需要编写任何代码，使用向导创建一个最简单的EVC 应用程序，并通过USB 同步连接的方式下载到开发板运行起来，这是我们的第一个基于EVC 的Hello, World 程序，在这里我们主要向您展示了最基本的开发步骤。

说明：使用基于GQ2440_SDK 的EVC 不需要安装庞大的PB5 平台即可进行应用开发。

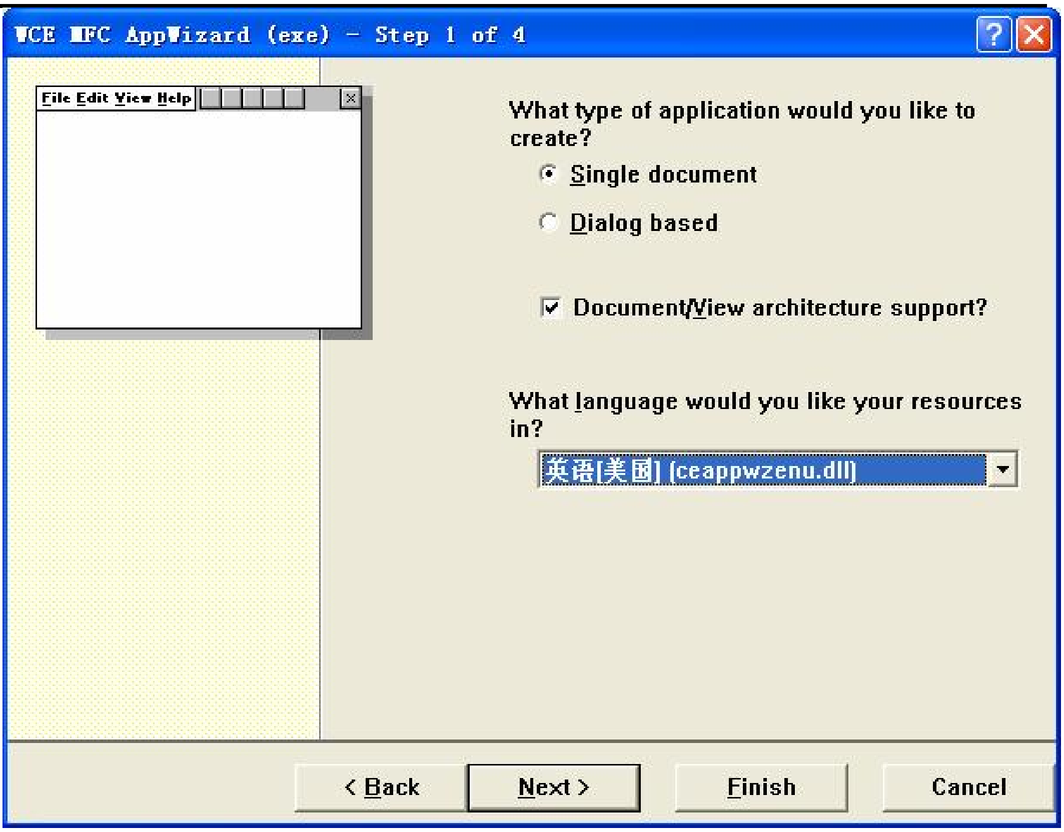
(1)打开运行EVC 集成开发环境，点【File】|【New…】如图



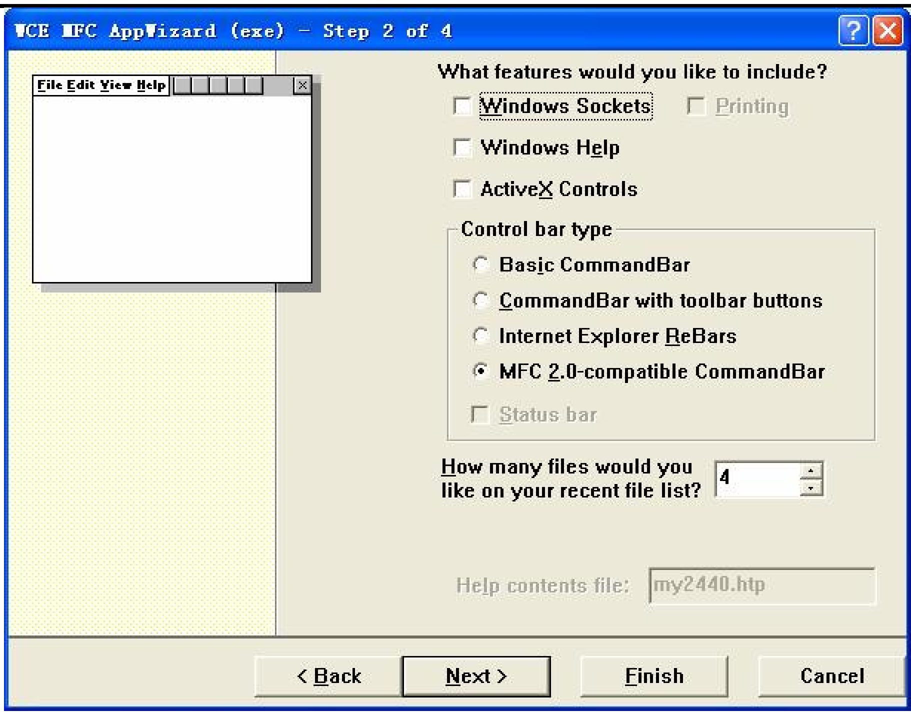
(2)出现程序向导窗口，选择WCE MFC AppWizard(exe)，在Project Name 中并输入项目名字，在 Location 中选择存放位置，如图，点“OK”继续



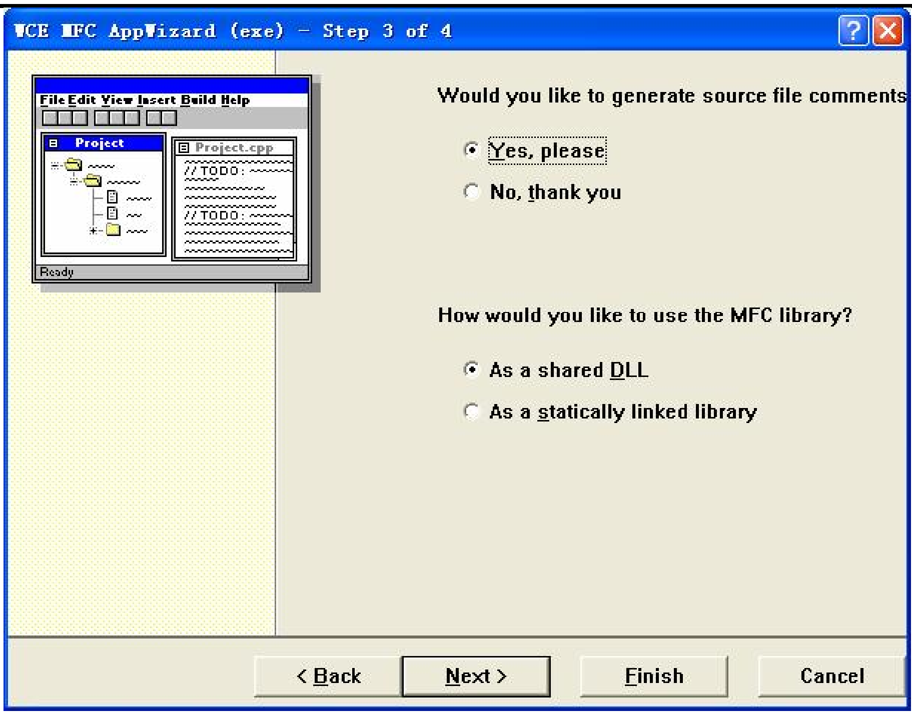
(3)出现如图窗口，选择语言设置为英语，其他为默认，点“Next”继续



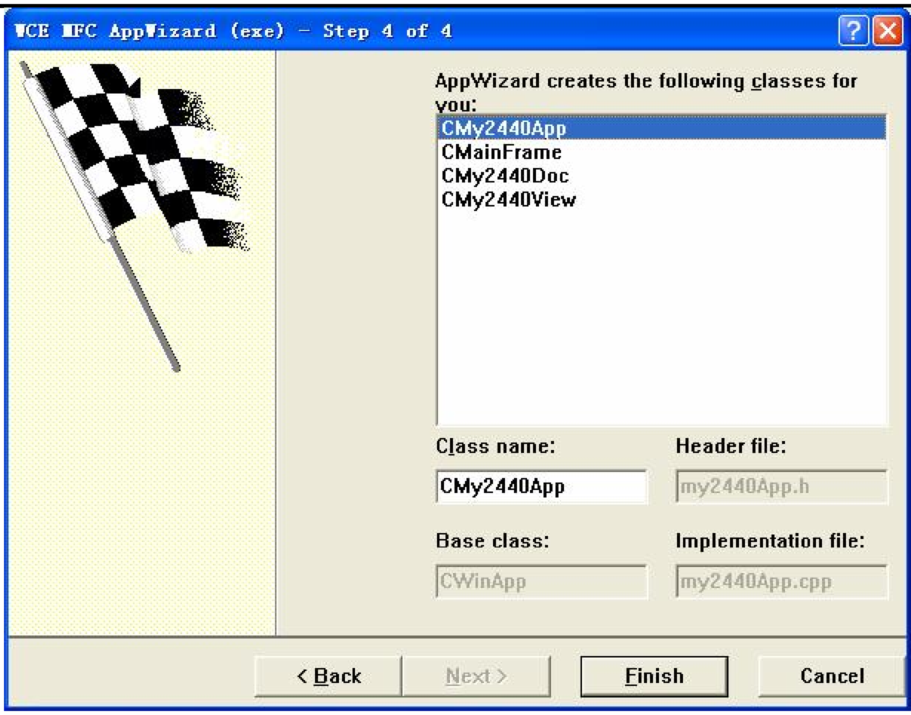
(4)出现如图窗口，所有设置保持默认，点“Next”继续



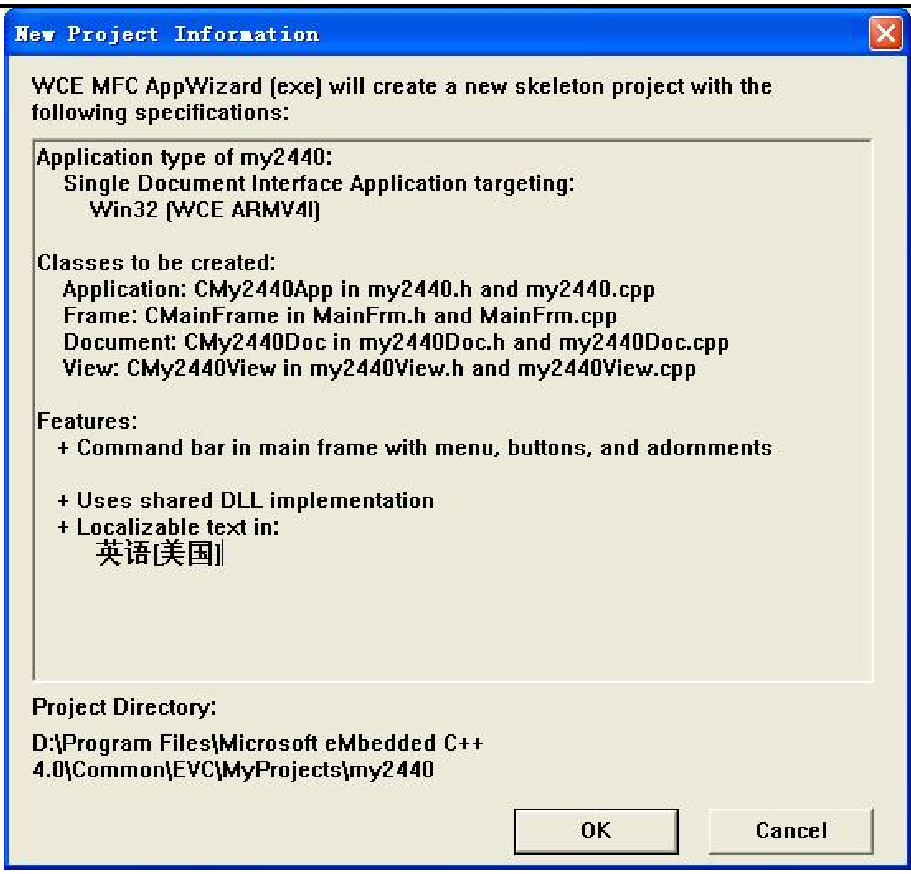
(5)出现如图界面，保持默认设置，点“Next”继续



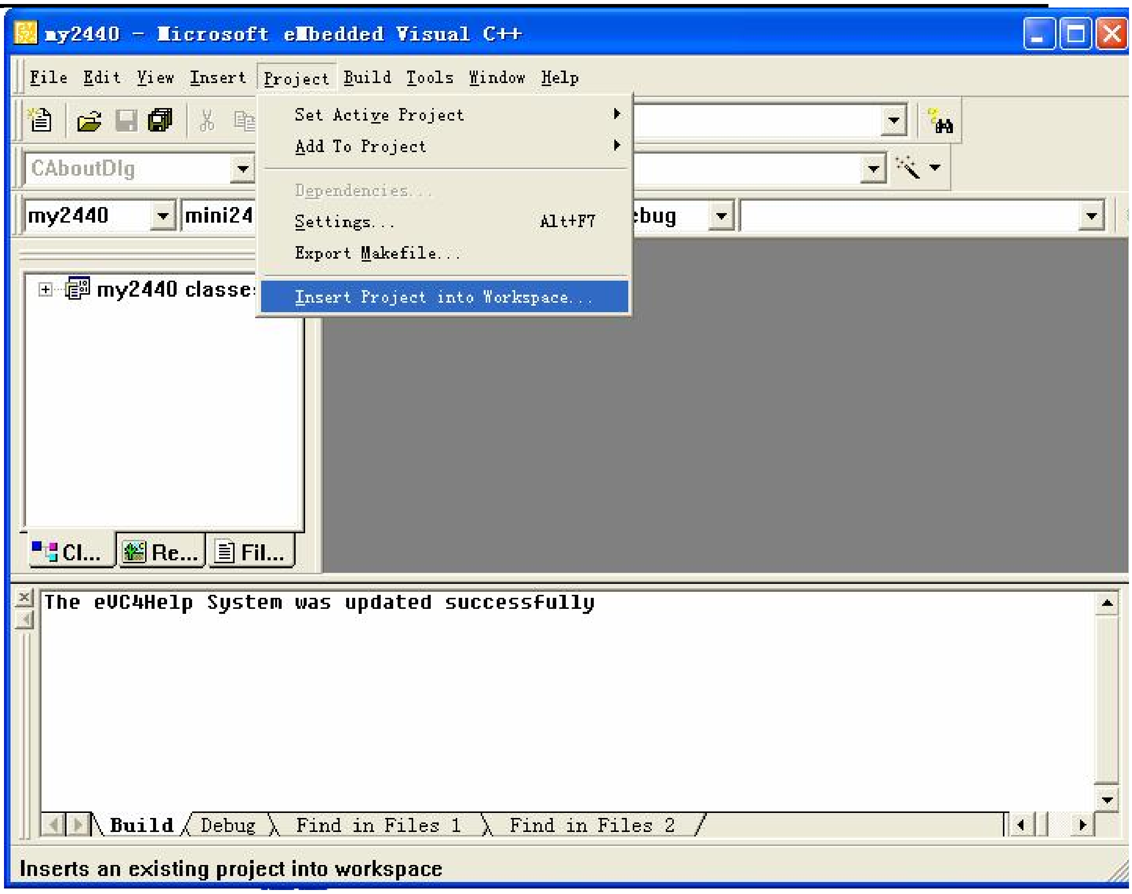
(6)出现如图界面，点“Finish”继续



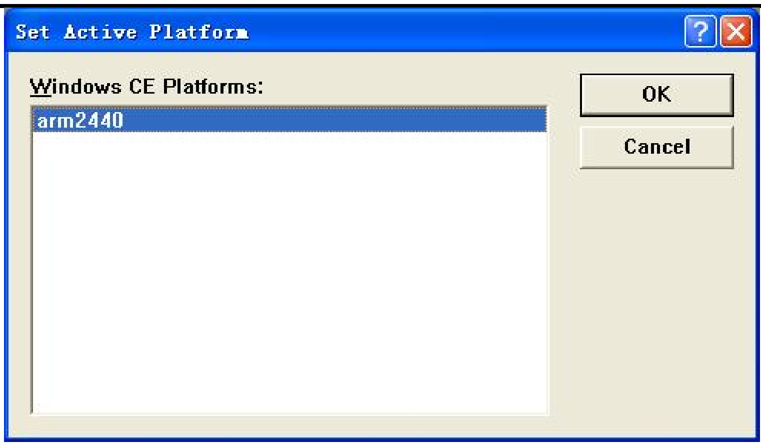
(7)出现New Project Information 窗口，点“OK”完成向导



(8)回到EVC 主界面，点【Build】|【Set Active Platform…】以设置目标板连接



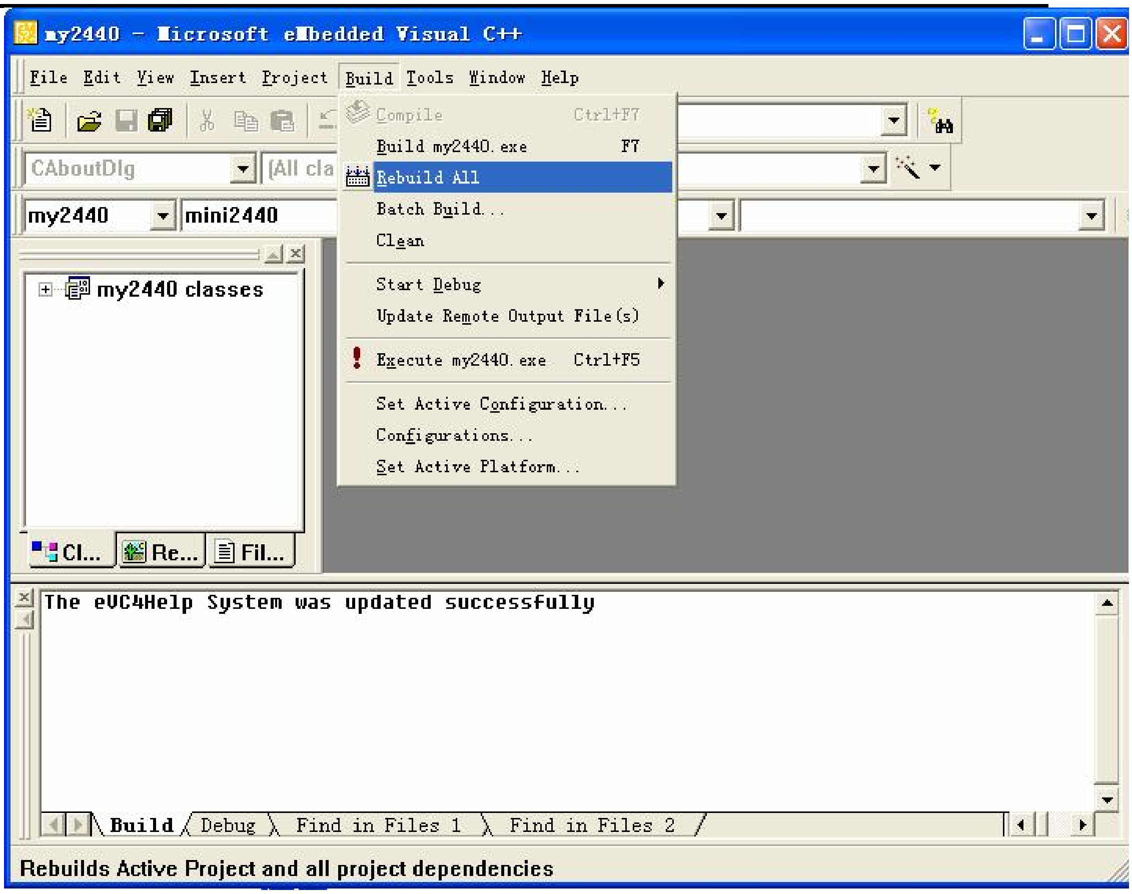
(9)因为我们只安装了导出的GQ2440_SDK，因此在这里只有一项选择，如图选择，点“OK”返回即可



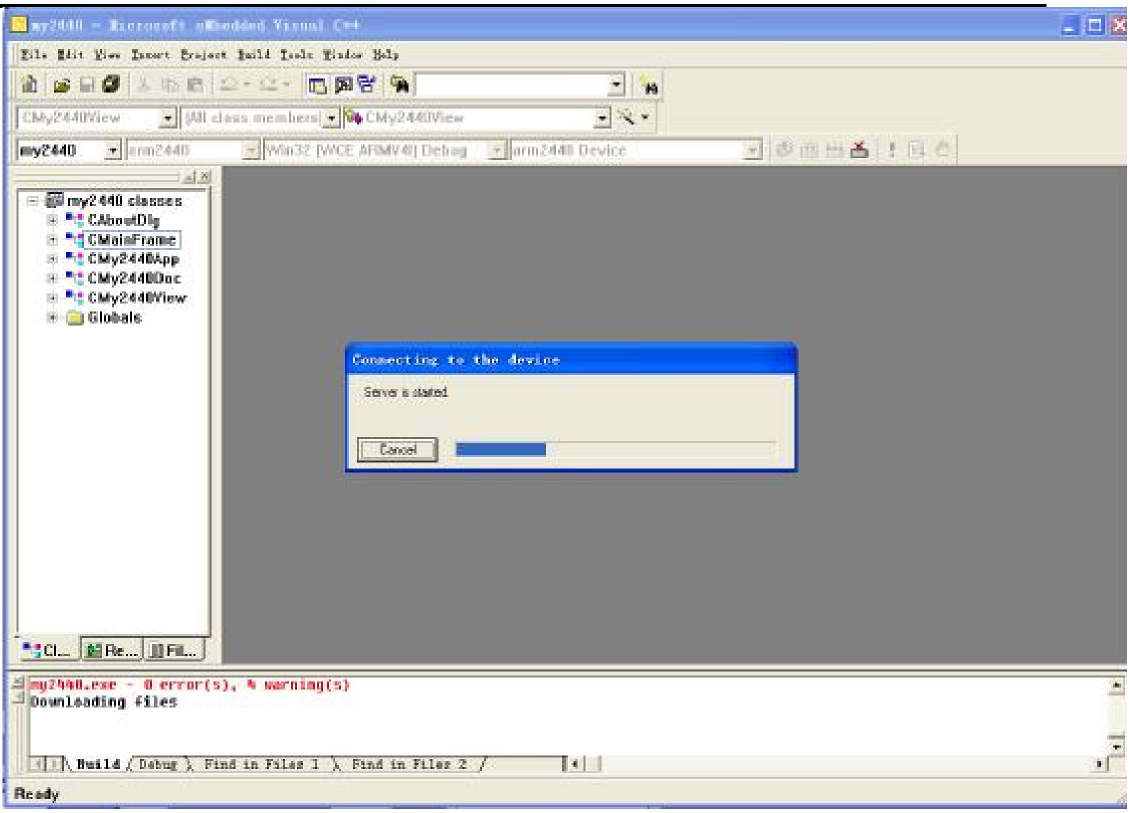
(10)确认PC 同步已经建立并连接正常(同时必须连接好网线)，如图



(11)在EVC 主界面，点【Build】|【Rebuild All】，如图，系统开始编译你所创建的工程文件：



(12)编译完毕，系统会自动下载下载编译好的可执行文件到开发板的根目录(打开“我的设备”就可以看到)，双击并运行它，如图：



my2440被自动下载到开发板:



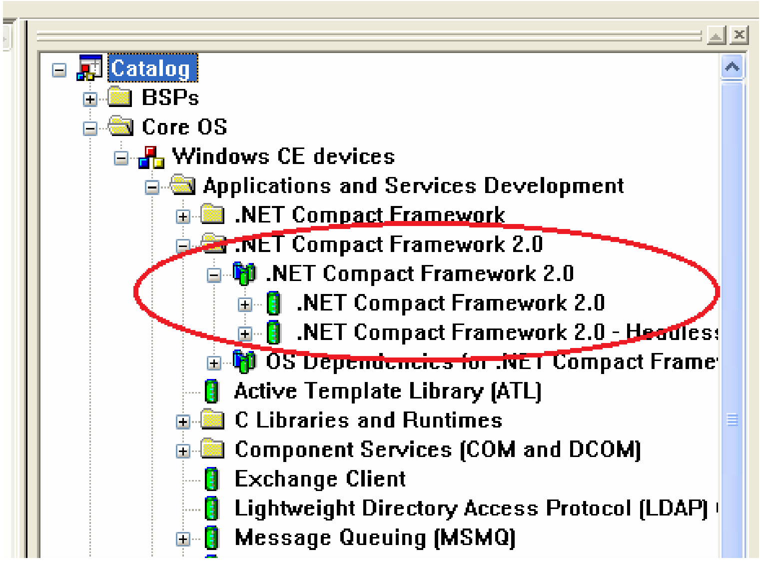
运行my2440程序界面:



9.4 创建 VS2005 应用程序，并编译下载到开发板运行

注意：必须要安装PB5 的补丁程序，才能在定制内核时出现.NET 2.0 选项，并使选择使用它才能支持VS2005/2008 应用程序。(缺省的内核稍写文件和示例工程已经添加了

该选项)如图:



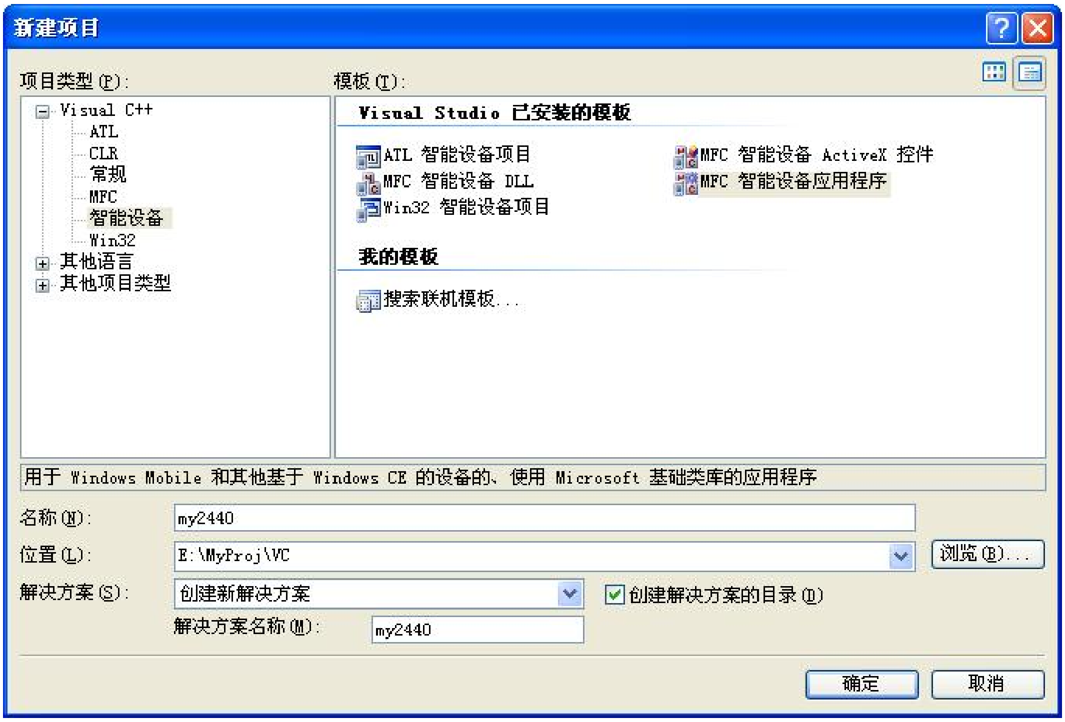
下面是使用 VS2005 的基本开发步骤:

9.4.1 创建项目

(1)打开运行VS2005，点菜单【文件】|【新建】|【项目】，如图:



(2)出现“新建项目”窗口，在“项目类型”一栏中点【Visual C++】|【智能设备】，在“模板”一栏中选择“MFC智能设备应用程序”，在“名称”一栏中输入“my2440”，其他采用缺省设置，点击【确定】继续：



(3)出现“MFC智能设备应用程序向导”，单击【下一步】：



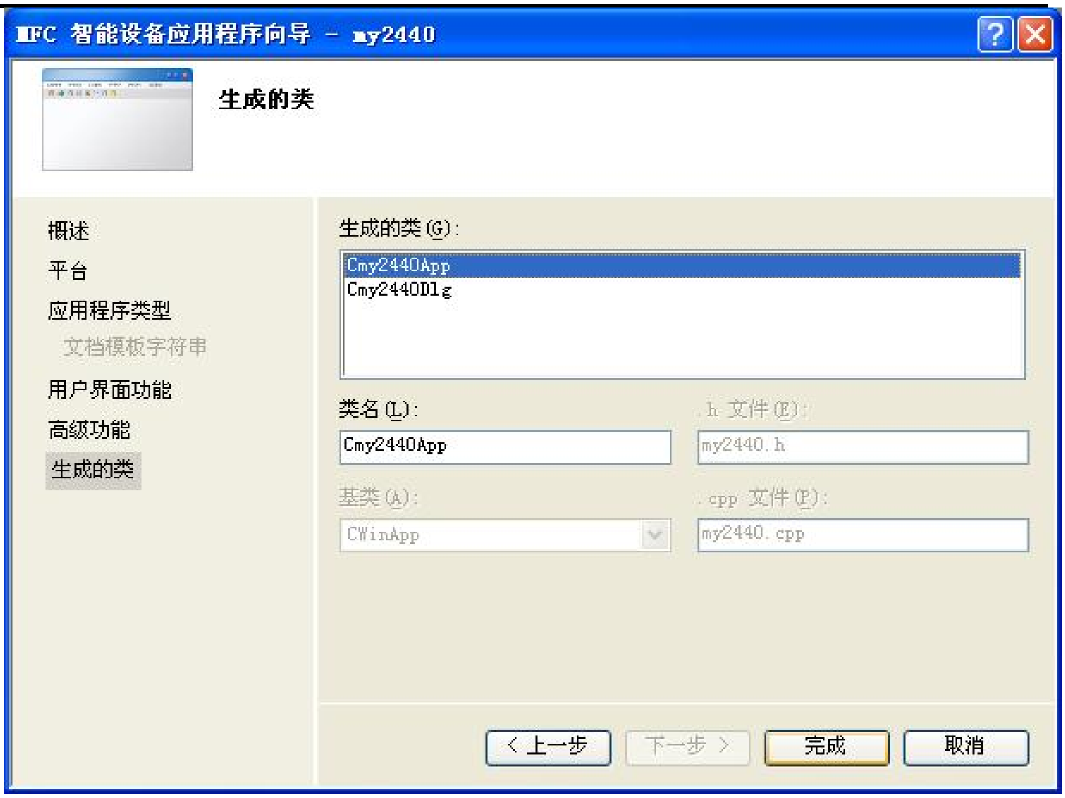
(4)在“平台”属性页中，将制作好的SDK——GQ2440添加到“选定的SDK”列表框中，单击【下一步】继续:



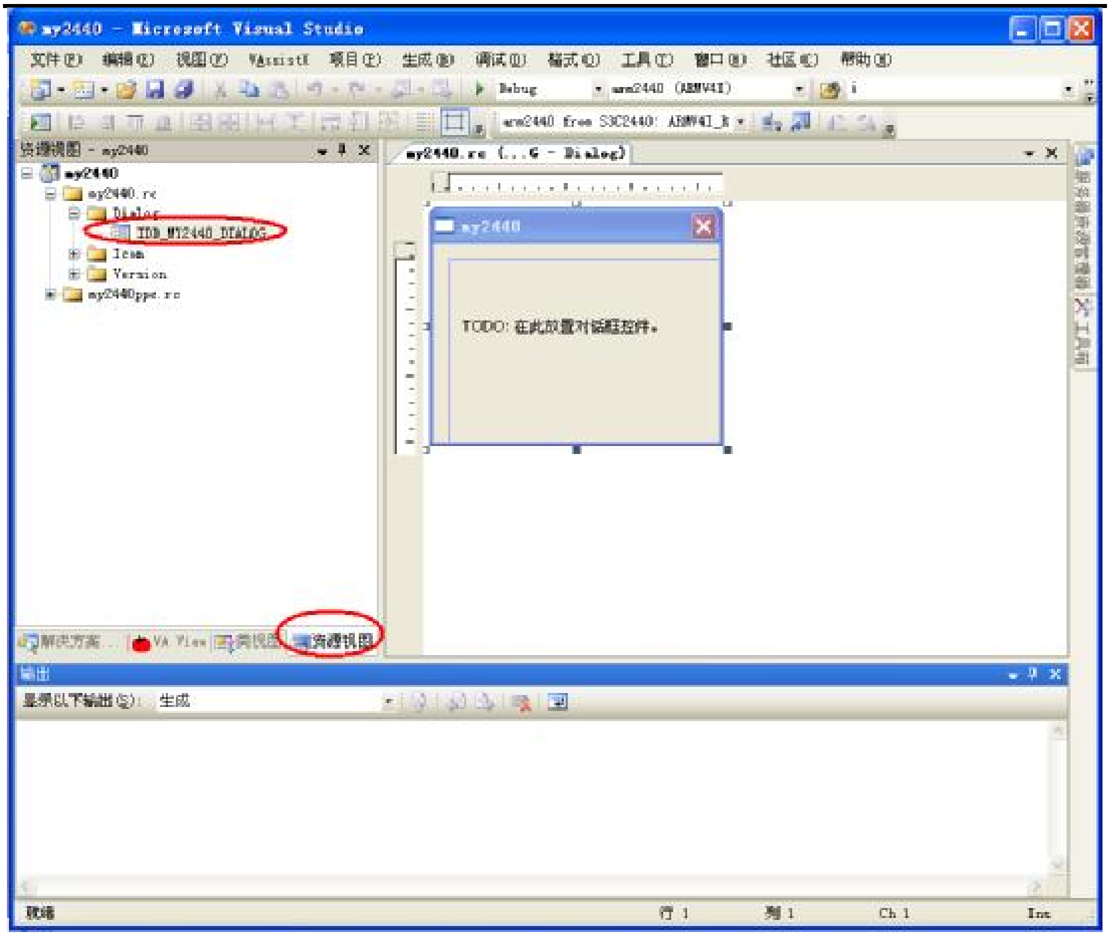
(5)在“应用程序类型”属性页中，选择“应用程序类型”为“基于对话框(D)”，选择“MFC的使用”为“在静态库中使用MFC(E)”，单击【下一步】继续：



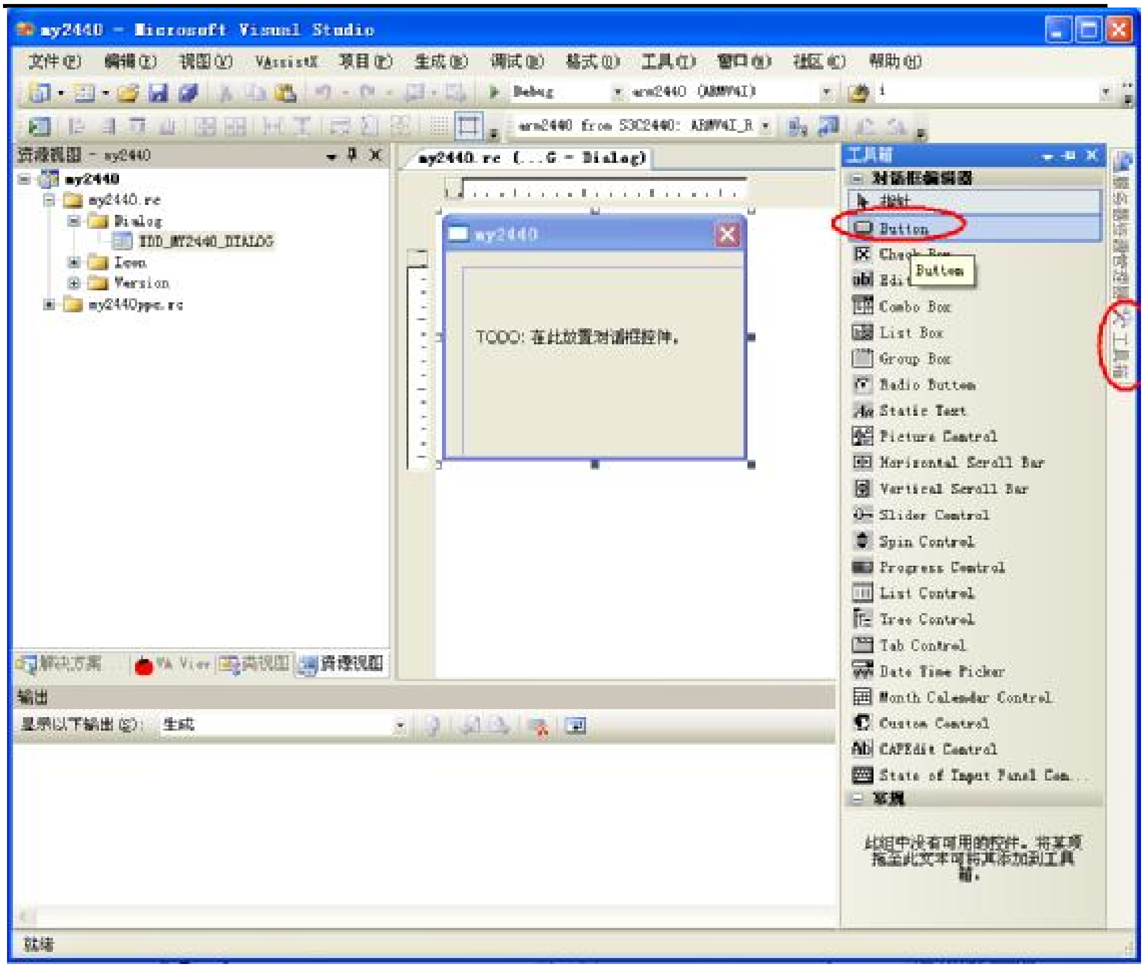
(6)以后的步骤只需选择默认选项即可，直至【完成】：



(7)通过【资源视图】|【my2440】|【my2440.rc】|【Dialog】|【IDD_MY2440_DIALOG】打开主对话框界面，如图所示:

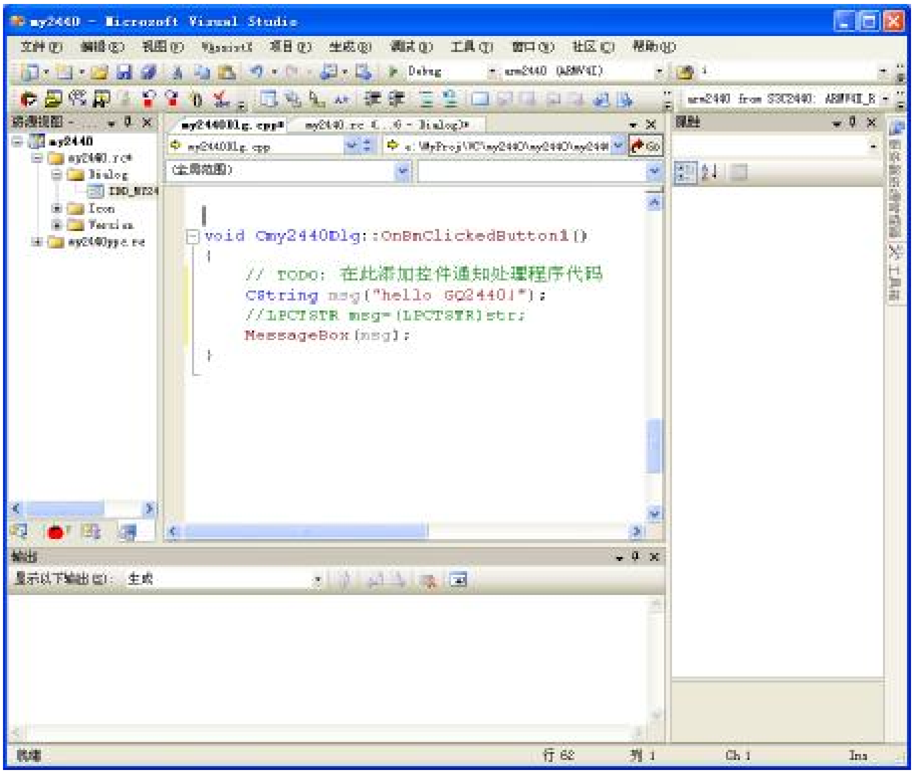


(8)在【工具箱】中选择Button控件，在对话框中绘制一个Button，如图所示:


my2440 - Microsoft Visual Studio
文件(F) 编辑(E) 视图(V) VAssistX 项目(P) 生成(B) 调试(D) 格式(O) 工具(T) 窗口(W) 社区(C) 帮助(H)
资源视图 - my2440
my2440
my2440.rc
Dialog
IDD_MY2440_DIALOG
Icon
Version
my2440ppc.rc
my2440.rc (...G - Dialog)
my2440
TODO: 在此放置对话框控件。
工具箱
对话框编辑器
指针
Button
Check Box
Edit Control
Combo Box
List Box
Group Box
Radio Button
Static Text
Picture Control
Horizontal Scroll Bar
Vertical Scroll Bar
Slider Control
Spin Control
Progress Control
List Control
Tree Control
Tab Control
Date Time Picker
Month Calendar Control
Custom Control
CAPEdit Control
State of Input Panel Con...
常规
此组中没有可用的控件。将某项拖至此文本可将其添加到工具箱。
输出
显示以下输出(S): 生成
就绪



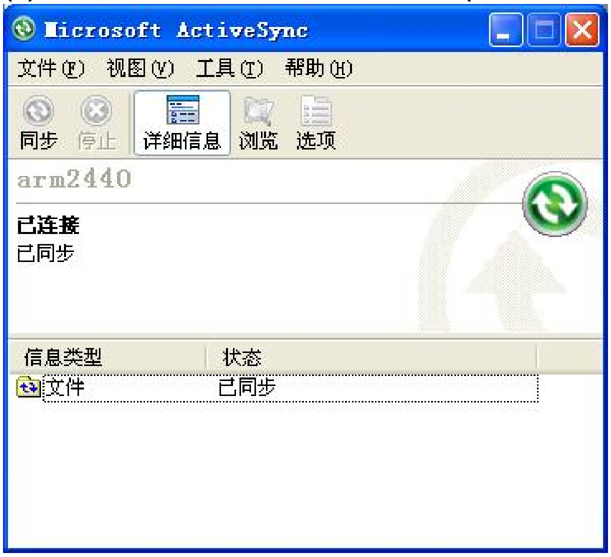
(9)双击新添加的控件“Button1”，可以进行按键回调函数编辑，添加如图所示代码:



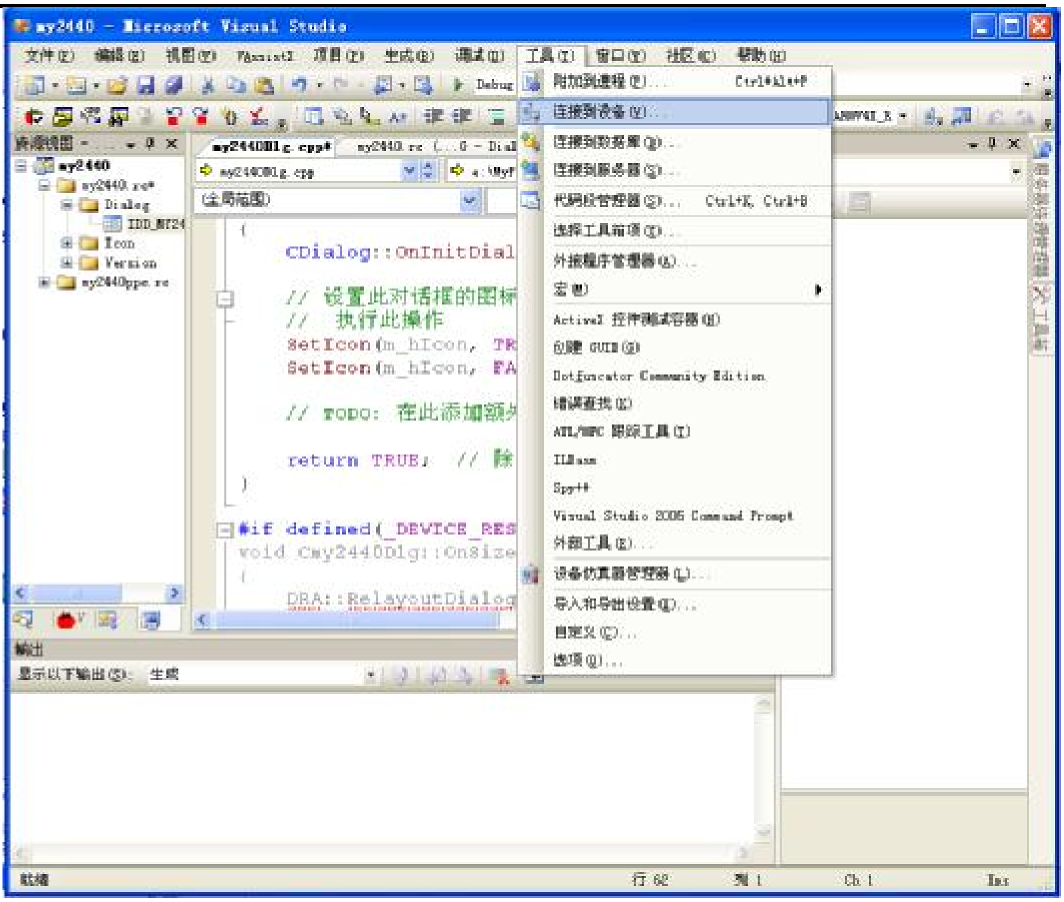
保存好以上创建的项目文件。

9.4.2 设置连接开发板

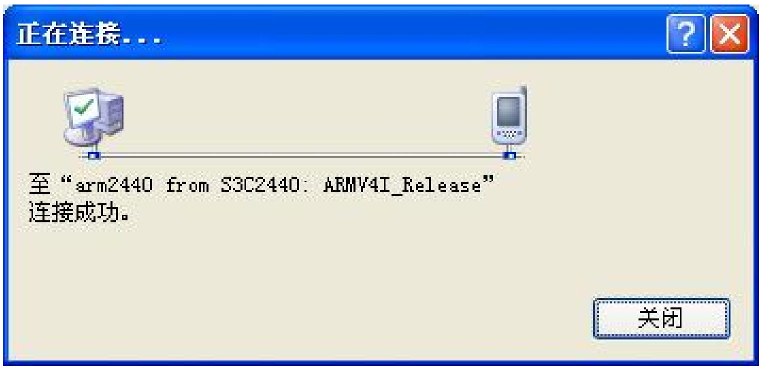
(1) 确认PC 同步已经建立并连接正常(可以不必连接网线)，如图



(2) 通过【工具】|【连接到设备】使pc与开发板建立连接，如图

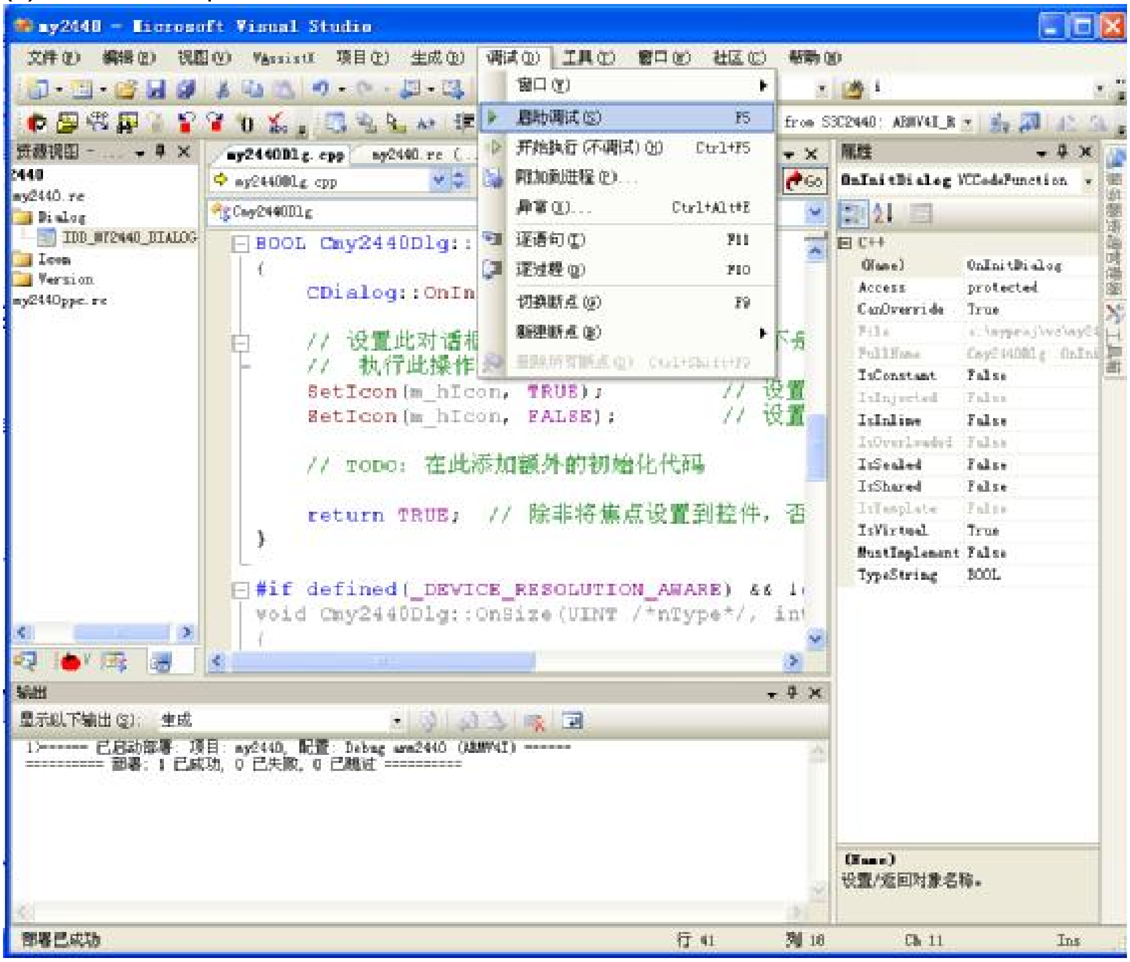


连接成功会出现如下界面:



9.4.3 设置连接开发板

(1)通过【调试】|【启动调试】或者直接按 F5 键开始调试过程:



(2)如果程序没有问题，等待一会它会直接下载到开发板并运行，点击一下“Button1”按钮，会弹出消息框，这正如我们在代码中的设置， 如图:





9.5 LED 驱动程序编写及测试示例

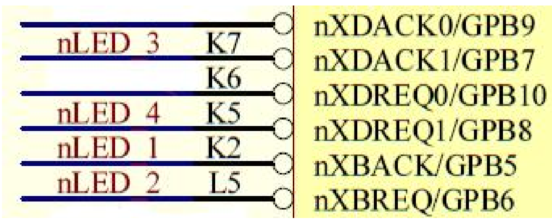
LED 驱动程序在BSP 中的位置：\GQ2440\Src\Drivers\LEDdriver
LED 驱动程序文件的名称：LEDDriver.cpp

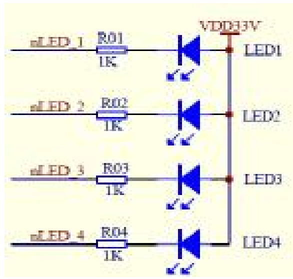
本小节以一个简单的 LED 驱动程序作为例子，介绍了Windows CE 下常

规流设备驱动程序编写的一般步骤和方法，用户可以通过更改其中的代码实现类似的其他IO 控制功能。

9.5.1 了解硬件连接

开发板上 4 个LED 引脚连接如图所示：





可见它们使用了 CPU 的GPB 端口，关于GPB 端口的描述及其使用在S3C2440 数据手册中有如下描述：

GPB 端口描述(见 S3C2440.pdf 第 276 页)：

GPB 对应寄存器：

Register	Address	R/W	Description	Reset Value
GPBCON	0x56000010	R/W	Configures the pins of port B	0x0
GPBDAT	0x56000014	R/W	The data register for port B	Undef.
GPBUP	0x56000018	R/W	Pull-up disable register for port B	0x0
Reserved	0x5600001c			

GPB 控制寄存器：

PBCON	Bit	Description
GPB8	[17:16]	00 = Input 01 = Output 10 = nXDREQ1 11 = Reserved
GPB7	[15:14]	00 = Input 01 = Output 10 = nXDACK1 11 = Reserved
GPB6	[13:12]	00 = Input 01 = Output 10 = nXBREQ 11 = reserved
GPB5	[11:10]	00 = Input 01 = Output 10 = nXBACK 11 = reserved

GPB 数据寄存器：

GPBDAT	Bit	Description
GPB[10:0]	[10:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

因此，要使用 GPB5~8，需要:

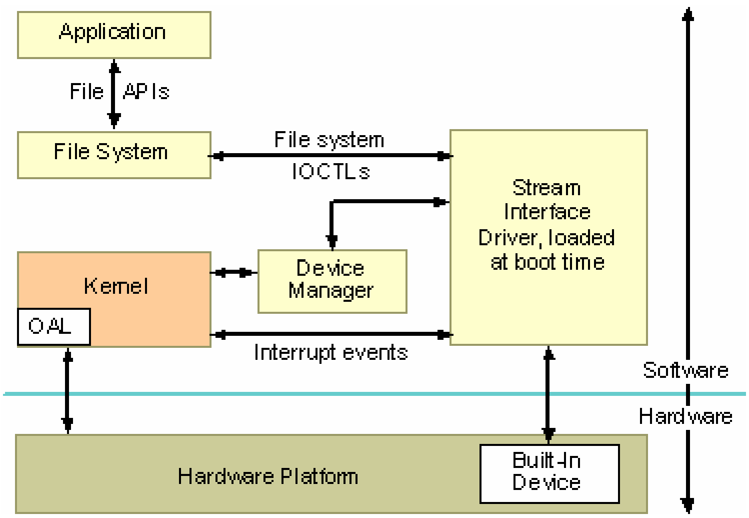
(1) 设置 GPBCON，以使引脚功能为 IO 功能输出

(2) 要点亮 LED, GPBDAT 对应的位要设置为 0

(3) 要熄灭 LED, GPBDAT 对应的位要设置为 1

9.5.2 编写 LED 流式驱动程序

流式驱动是 WinCE 驱动程序的一种常规方式，应用程序通过文件系统，透过 DeviceManager 以访问文件的形式访问驱动程序，调用 IOCTL 向驱动程序下达指令。所有的流式驱动程序都需实现一组统一的接口。



流式驱动程序接口函数(摘自 PB5 帮助文档)

Programming element	Description
XXX_Close (Device Manager)	This function closes the device context identified by *hOpenContext*.

	This function is required to access the device with CreateFile. If you implement **XXX_Close**, you must implement **XXX_Open**.
XXX_Deinit (Device Manager)	This function de-initializes a device. It is called by Device Manager. This function is required by drivers loaded by ActivateDeviceEx, ActivateDevice, or RegisterDevice.
XXX_Init (Device Manager)	This function initializes a device. It is called by Device Manager. This function is required by drivers loaded by **ActivateDeviceEx**, **ActivateDevice**, or **RegisterDevice**.
XXX_IOControl (Device Manager)	This function sends a command to a device. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open**and **XXX_Close**.
XXX_Open (Device Manager)	This function opens a device for reading, writing, or both. An application indirectly invokes this function when it calls **CreateFile** to obtain a handle to a device. This function is required to access the device with **CreateFile**.
XXX_PowerDown (Device Manager)	Optional. This function ends power to the device. It is useful only with devices that can be shut off under software control.
XXX_PowerUp (Device Manager)	Optional. This function restores power to a device.
XXX_PreClose (Device Manager)	Optional. This function marks the closing handle as invalid and wakes any sleeping threads.
XXX_PreDeinit (Device Manager)	This function marks the device instance as invalid and wakes sleeping threads. This function is required if the **XXX_PreClose**function is implemented.
XXX_Read (Device Manager)	This function reads data from the device identified by the open context. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open**and **XXX_Close**.
XXX_Seek (Device Manager)	This function moves the data pointer in the device. This function might or might not be required, depending on the device

	capabilities that the driver exposes. This function requires an implementation of **XXX_Open** and **XXX_Close**.
XXX_Write (Device Manager)	This function writes data to the device. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open**and **XXX_Close**.

据此，我们要实现的 LED 驱动主要源代码如下：

```
#include <windows.h>
//#include <ceddk.h>
#include <nkintr.h>
#include <pm.h>
#include "pmplatform.h"
#include "Pkfuncs.h"
#include "s2440.h"


#define IO_CTL_LED_1_ON 0x01
#define IO_CTL_LED_2_ON 0x02
#define IO_CTL_LED_3_ON 0x03
#define IO_CTL_LED_4_ON 0x04
#define IO_CTL_LED_ALL_ON 0x05
#define IO_CTL_LED_1_OFF 0x06
#define IO_CTL_LED_2_OFF 0x07
#define IO_CTL_LED_3_OFF 0x08
#define IO_CTL_LED_4_OFF 0x09
#define IO_CTL_LED_ALL_OFF 0x0a

volatile IOPreg   *s2440IOP = (IOPreg *)IOP_BASE;
```

```
volatile INTreg   *s2440INT = (INTreg *)INT_BASE;

BOOL mInitialized;
void Virtual_Alloc();                                    // Virtual allocation

void Virtual_Alloc()
{

    // GPIO Virtual alloc
    s2440IOP = (volatile IOPreg *) VirtualAlloc(0,sizeof(IOPreg),MEM_RESERVE, PAGE_NOACCESS);
    if(s2440IOP == NULL) {
        RETAILMSG(1,(TEXT("For s2440IOP: VirtualAlloc faiLED!\r\n")));
    }
    else {
        if(!VirtualCopy((PVOID)s2440IOP,(PVOID)(IOP_BASE),sizeof(IOPreg),PAGE_READWRITE       |
PAGE_NOCACHE )) {
            RETAILMSG(1,(TEXT("For s2440IOP: VirtualCopy faiLED!\r\n")));
        }
    }



}

BOOL WINAPI
DllEntry(HANDLE       hinstDLL,
            DWORD dwReason,
            LPVOID /* lpvReserved */)
{
    switch(dwReason)
    {
    case DLL_PROCESS_ATTACH:
        DEBUGREGISTER((HINSTANCE)hinstDLL);
```

```
            return TRUE;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
        case DLL_PROCESS_DETACH:
            break;
#ifdef UNDER_CE
        case DLL_PROCESS_EXITING:
            break;
        case DLL_SYSTEM_STARTED:
            break;
#endif
        }

        return TRUE;
}

BOOL LED_Deinit(DWORD hDeviceContext)
{
        BOOL bRet = TRUE;

        RETAILMSG(1,(TEXT("USERLED: LED_Deinit\r\n")));

        return TRUE;
}

BOOL LEDGpioInit()
{
        RETAILMSG(1,(TEXT("LED_Gpio_Setting----\r\n")));
        s2440IOP->rGPBCON    = (s2440IOP->rGPBCON    &~(3 << 10)) | (1<< 10);   // GPB5 == OUTPUT.
        s2440IOP->rGPBCON    = (s2440IOP->rGPBCON    &~(3 << 12)) | (1<< 12);   // GPB6 == OUTPUT.
```

```
        s2440IOP->rGPBCON    = (s2440IOP->rGPBCON    &~(3 << 14)) | (1<< 14);   // GPB7 == OUTPUT.
        s2440IOP->rGPBCON    = (s2440IOP->rGPBCON    &~(3 << 16)) | (1<< 16);   // GPB8 == OUTPUT.
        return TRUE;
}

DWORD LED_Init(DWORD dwContext)
{

        RETAILMSG(1,(TEXT("LED_Init----\r\n")));

        // 1. Virtual Alloc
        Virtual_Alloc();

        LEDGpioInit();



        mInitialized = TRUE;
        return TRUE;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
BOOL LED_IOControl(DWORD hOpenContext,
                                DWORD dwCode,
                                PBYTE pBufIn,
                                DWORD dwLenIn,
                                PBYTE pBufOut,
                                DWORD dwLenOut,
                                PDWORD pdwActualOut)
{
        LEDGpioInit();
        switch(dwCode)
```

```
{
case IO_CTL_LED_1_ON:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<5);
	break;
case IO_CTL_LED_2_ON:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<6);
	break;
case IO_CTL_LED_3_ON:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<7);
	break;
case IO_CTL_LED_4_ON:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<8);
	break;
case IO_CTL_LED_ALL_ON:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0xF<<5);
	break;
case IO_CTL_LED_1_OFF:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<5);
	break;
case IO_CTL_LED_2_OFF:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<6);
	break;
case IO_CTL_LED_3_OFF:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<7);
	break;
case IO_CTL_LED_4_OFF:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<8);
	break;
case IO_CTL_LED_ALL_OFF:
	s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0xF<<5);
	break;
default:
```

```
                break;
        }

        RETAILMSG(1,(TEXT("LED:Ioctl code = 0x%x\r\n"), dwCode));
        return TRUE;
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
DWORD LED_Open(DWORD hDeviceContext, DWORD AccessCode, DWORD ShareMode)
{
        RETAILMSG(1,(TEXT("USERLED: LED_Open\r\n")));
        return TRUE;
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
BOOL LED_Close(DWORD hOpenContext)
{
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&(0xF<<5);
        RETAILMSG(1,(TEXT("USERLED: LED_Close\r\n")));
        return TRUE;
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
void LED_PowerDown(DWORD hDeviceContext)
{
        RETAILMSG(1,(TEXT("USERLED: LED_PowerDown\r\n")));

        //RETAILMSG(1,(TEXT("CAMERA: LED_PowerDown, m_Dx = D%u, init %d \r\n"), m_Dx, mInitialized));
}
```

```
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
void LED_PowerUp(DWORD hDeviceContext)
{
	RETAILMSG(1,(TEXT("USERLED: LED_PowerUp\r\n")));

}

//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
DWORD LED_Read(DWORD hOpenContext, LPVOID pBuffer, DWORD Count)
{
	RETAILMSG(1,(TEXT("USERLED: LED_Read\r\n")));
	return TRUE;
}

//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
DWORD LED_Seek(DWORD hOpenContext, long Amount, DWORD Type)
{
	RETAILMSG(1,(TEXT("USERLED: LED_Seek\r\n")));
	return 0;
}

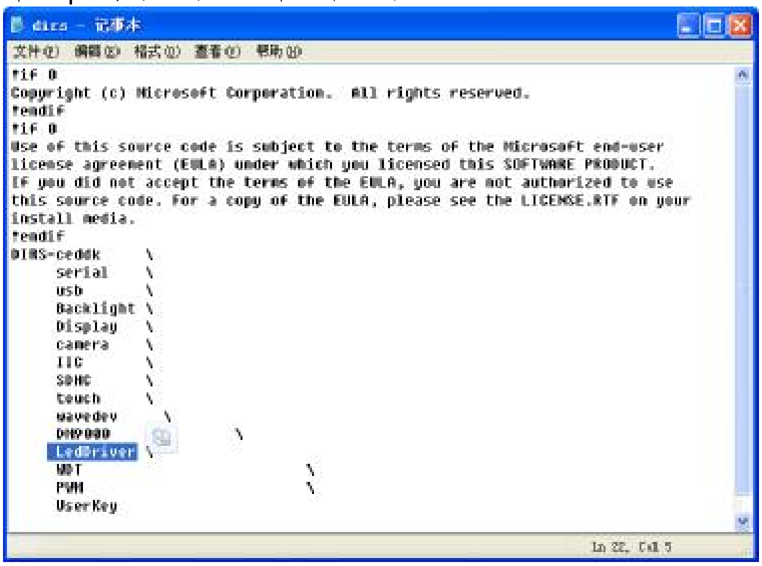
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
DWORD LED_Write(DWORD hOpenContext, LPCVOID pSourceBytes, DWORD NumberOfBytes)
{
	RETAILMSG(1,(TEXT("USERLED: LED_Write\r\n")));
	return 0;
}
```

9.5.3 把 LED 驱动程序添加到 BSP 中以编译

LED 驱动程序必须要编译到内核中才能使用，要把 LED 驱动程序编译到内核中，需要做如下步骤:

(1) 建立 LED 驱动程序目录

在 GQ2440\DRIVERS 下建立 LEDdriver 目录，并在 dirs 文件中加入此目录，使系统编译 bsp 的时候可以编译这个文件。



```
dirs - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#if 0
Copyright (c) Microsoft Corporation.  All rights reserved.
#endif
#if 0
Use of this source code is subject to the terms of the Microsoft end-user
license agreement (EULA) under which you licensed this SOFTWARE PRODUCT.
If you did not accept the terms of the EULA, you are not authorized to use
this source code. For a copy of the EULA, please see the LICENSE.RTF on your
install media.
#endif
DIRS=ceddk     \
     serial    \
     usb       \
     Backlight \
     Display   \
     camera    \
     IIC       \
     SDHC      \
     touch     \
     wavedev     \
     DM9000          \
     LedDriver \
     WDT                  \
     PWM                  \
     UserKey
Ln 22, Col 5
```

(2)为 LED 驱动创建 Makefile 文件

可以参考其他驱动程序示例创建 Makefile 文件。

在 GQ2440\DRIVERS\LEDriver\目录中建立 makefile 文件，内容如下:

```
#
# DO NOT EDIT THIS FILE!!! Edit .\sources. if you want to add a new source
# file to this component. This file merely indirects to the real make file
# that is shared by all the components of Peg
#
    !INCLUDE $(_MAKEENVROOT)\makefile.def
```

(3)为 LED 驱动创建 source 文件

可以参考其他驱动程序实例创建 source 文件。

在 GQ2440\DRIVERS\LEDriver\目录中建立 source 文件，内容如下：

```
!if 0
File:    sources

Author: jeffmi

        Copyright (c) 1995-2002 Microsoft Corporation.   All rights reserved.
        !endif

        RELEASETYPE=PLATFORM
        TARGETNAME=LEDDriver
        TARGETTYPE=DYNLINK
        DLLENTRY=DllEntry

        TARGETLIBS= \
        $(_COMMONSDKROOT)\lib\$(_CPUINDPATH)\coredll.lib \

        MSC_WARNING_LEVEL = $(MSC_WARNING_LEVEL) /W3 /WX

        INCLUDES= \
        $(_TARGETPLATROOT)\inc; \
        $(_COMMONOAKROOT)\inc; \

    $(_PUBLICROOT)\common\oak\inc;$(_PUBLICROOT)\common\sdk\inc;$(_PUBLICRO
OT)\common\ddk\inc;   \
        ..\..\inc

        SOURCES= \
        leddriver.cpp \
```

(4)编写 leddriver.def 导出 Dll 符号

可以参考其他驱动程序示例创建 leddriver.def 文件。

(5)把 LED 驱动加入内核

```
LIBRARY userLED
EXPORTS
LED_Close
LED_Deinit
LED_Init
LED_IOControl
LED_Open
LED_PowerDown
LED_PowerUp
LED_Read
LED_Seek
LED_Write
```

在配置文件 platform.bib 中加入以下内容:

```
;leddriver
leddriver.dll $(_FLATRELEASEDIR)\leddriver.dll NK SH
```

(6)把 LED 驱动加入注册表

在注册表文件 platform.reg 中加入以下内容:

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\LEDdriver]

"Prefix"="LED"

"Dll"="LEDdriver.dll"

(7)重新编译内核

重新编译内核，点 PB5 的主菜单【Build】|【OSSysgen】即可，这样就可以生成包含以上 LED 驱动的内核映象文件 NK.bin 和 NK.nb0 了。

9.5.4 编写并编译 LED 测试应用程序

示例文件在光盘中的位置：\WinCE 源代码和软件\WindowsCE5.0\WindowsCE 应用程序示例代码\LEDtest
示例文件所使用的编译器：eMbedded Visual C++ 4.0 + SP4 补丁 + GQ2440_SDK
SP4 补丁在光盘中的位置：\Embedded VisualC++\SP\evc4sp4\DISK1
GQ2440_SDK 安装文件在光盘中的位置：\WinCE 源代码和软件\WindowsCE5.0\SDK

要在应用程序中调用驱动程序，IO Control 是最常用的方法，它的一般流程如下：
在应用程序中：

```
HANDLE
leddriver=CreateFile(L"LED1:",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_E
XISTING,0,NULL );
ReadFile(leddriver，……………….)
DeviceIoControl(leddriver,IO_CTL_LED_CLEAR,NULL,0,NULL,0,NULL,NULL);
```

在驱动程序中：
当 CreateFile 被调用时，调用 XXX_Open
当 ReadFile 被调用时，调用 XXX_Read
当 DeviceIoControl 被调用时，调用 XXX_IOContrl,根据 IO_CTL_XXX 的数值，决定执行的操作。
在 LED 驱动中，实现 10 个 IOControl 分支，处理 10 个命令请求，对应点亮 1~4 号灯，点亮所有 LED，关闭 1~4 灯等，关闭所有 LED 灯。
LED 测试程序的主要代码如下：

```
// LEDTESTDlg.cpp : implementation file
```

```
//

#include "stdafx.h"
#include "LEDTEST.h"
#include "LEDTESTDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define IO_CTL_LED_1_ON 0x01
#define IO_CTL_LED_2_ON 0x02
#define IO_CTL_LED_3_ON 0x03
#define IO_CTL_LED_4_ON 0x04
#define IO_CTL_LED_ALL_ON 0x05
#define IO_CTL_LED_1_OFF 0x06
#define IO_CTL_LED_2_OFF 0x07
#define IO_CTL_LED_3_OFF 0x08
#define IO_CTL_LED_4_OFF 0x09
#define IO_CTL_LED_ALL_OFF 0x0a

/////////////////////////////////////////////////////////////////////////////
// CLEDTESTDlg dialog

CLEDTESTDlg::CLEDTESTDlg(CWnd* pParent /*=NULL*/)
: CDialog(CLEDTESTDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CLEDTESTDlg)
	// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
```

```
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CLEDTESTDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CLEDTESTDlg)
	// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CLEDTESTDlg, CDialog)
	//{{AFX_MSG_MAP(CLEDTESTDlg)
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
	ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
	ON_BN_CLICKED(IDC_BUTTON4, OnButton4)
	ON_BN_CLICKED(IDC_BUTTON5, OnButton5)
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_BUTTON6, OnButton6)
	ON_BN_CLICKED(IDC_BUTTON7, OnButton7)
	ON_BN_CLICKED(IDC_BUTTON8, OnButton8)
	ON_BN_CLICKED(IDC_BUTTON9, OnButton9)
	ON_BN_CLICKED(IDC_BUTTON10, OnButton10)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLEDTESTDlg message handlers

BOOL CLEDTESTDlg::OnInitDialog()
{
```

```
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	
	CenterWindow(GetDesktopWindow());	// center to the hpc screen


	leddriver=CreateFile(L"LED1:",GENERIC_READ | GENERIC_WRITE, 0,NULL,OPEN_EXISTING,0,NULL );

	if(!leddriver)
		MessageBox(L"ERROR OPENING LED!");

	return TRUE;  // return TRUE  unless you set the focus to a control
}



BOOL CLEDTESTDlg::PreTranslateMessage(MSG* pMsg) 
{
	// TODO: Add your specialized code here and/or call the base class
	return CDialog::PreTranslateMessage(pMsg);
}

void CLEDTESTDlg::OnButton1() 
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_1_ON, NULL,0,NULL,0,NULL,NULL);
```

```
}

void CLEDTESTDlg::OnButton2()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_2_ON,NULL,0,NULL,0,NULL,NULL);
}

void CLEDTESTDlg::OnButton3()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_3_ON,NULL,0,NULL,0,NULL,NULL);
}

void CLEDTESTDlg::OnButton4()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_4_ON,NULL,0,NULL,0,NULL,NULL);
}

BOOL CLEDTESTDlg::DestroyWindow()
{
	// TODO: Add your specialized code here and/or call the base class
	DeviceIoControl(leddriver,IO_CTL_LED_ALL_OFF,NULL,0,NULL,0,NULL,NULL);
	return CDialog::DestroyWindow();
}

void CLEDTESTDlg::OnButton5()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_ALL_ON,NULL,0,NULL,0,NULL,NULL);
}
```

```
void CLEDTESTDlg::OnClose()
{
	// TODO: Add your message handler code here and/or call default
	DeviceIoControl(leddriver,IO_CTL_LED_ALL_OFF,NULL,0,NULL,0,NULL,NULL);
	CDialog::OnClose();
}

void CLEDTESTDlg::OnButton6()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_1_OFF,NULL,0,NULL,0,NULL,NULL);
}

//DEL void CLEDTESTDlg::OnDoubleclickedButton6()
//DEL {
//DEL 	// TODO: Add your control notification handler code here
//DEL
//DEL }

void CLEDTESTDlg::OnButton7()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_2_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CLEDTESTDlg::OnButton8()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_3_OFF,NULL,0,NULL,0,NULL,NULL);
}
```

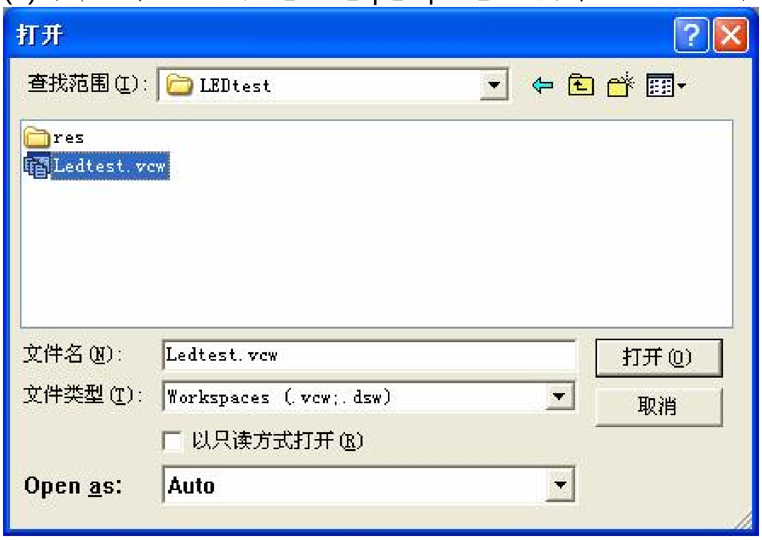
```
void CLEDTESTDlg::OnButton9()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_4_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CLEDTESTDlg::OnButton10()
{
	// TODO: Add your control notification handler code here
	DeviceIoControl(leddriver,IO_CTL_LED_ALL_OFF,NULL,0,NULL,0,NULL,NULL);
}
```

要编译光盘中测 LEDtest 测试程序，步骤如下:

(1)把 LEDtest 复制到硬盘的某个目录，比如 D:\Work，去掉其只读属性。

(2)打开运行 EVC，点【File】|【Open】，找到 LEDtest 工程文件，如图:



(3)参考 9.4 一节的方法先设置好开发板连接(ActiveSync 同步和网络都必须连接正常)，点菜单【Build】|【Rebuild All】开始重新编译 LEDtest.exe 程序，编译完毕，EVC 会自动连接下载到开发板的**我的设备**目录，同时在 D:\work\lcdtest\ARMV4IRel 目录下生成

LCDtest.exe 可执行程序，点击运行 LCDtest 测试程序，如图，可以按照提示进行控制开发板上的 LED。如图：



9.5.5 把 LED 测试程序添加到内核，并建立桌面快捷方式

为了把上一小节编译出的 LEDtest.exe 可执行程序永久写入到 NK.bin/NK.nb0 中，以便每次开机时都能从桌面的快捷方式中执行它，我们采用以下方法步骤：

(1)把 LEDtest.exe 复制到 GQ2440\FILES 目录中。

(2)使用 Windows 自带的记事本程序创建 LEDtest.lnk 快捷方式文件:

.lnk 文件其实是一个文本文件，它包含用于链接目标的命令行以及命令行的长度，其格式为“<length>#<command line>”，其中 length 是#后所有字符的个数，在此，LEDtest.lnk 的内容如下:

20#\Windows\LEDtest.exe

注意：当把 LEDtest.lnk 后缀改为.lnk 后，使用记事本一般就无法打开了。

我们把这个文件也放入 GQ2440\FILES 目录中。

(3)打开 platform.bib 文件，添加如下内容:

LEDtest.exe $(_FLATRELEASEDIR)\QQtest.exe NK U

LEDtest.lnk $(_FLATRELEASEDIR)\QQtest.lnk NK U

这样，执行 SYSGEN 的时候会把这两个文件加入到内核中，最后他们会存在于开发板的\Windows 目录中。

(4)打开 platform.dat，加入以下内容:

Directory("\windows\桌面"):-File("LED 测试.lnk","\windows\LEDtest.lnk")

这将会在桌面出现名称位“LED 测试”的快捷方式，它是 LEDtest.lnk 的一个拷贝，其内容和 LEDtest.lnk 是一样的。

(5)最后执行菜单【Builder】|【Sysgen】，生成 NK.bin 和 NK.nb0，把它们烧写或者下载到开发板启动后，就会在桌面看到“LED 测试”快捷方式了。